2017-12

Mobile robot navigation and obstacle avoidance in unstructured outdoor environments

Hargadine, Calvin S.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MOBILE ROBOT NAVIGATION AND OBSTACLE AVOIDANCE IN UNSTRUCTURED OUTDOOR ENVIRONMENTS

by

Calvin S. Hargadine

December 2017

Thesis Advisor: Xiaoping Yun
Co-Advisor: James Calusdian

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE MOBILE ROBOT NAVIGATION AND OBSTACLE AVOIDANCE IN UNSTRUCTURED OUTDOOR ENVIRONMENTS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Calvin S. Hargadine			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The ability to detect, characterize, and avoid obstacles is a critical requirement for autonomous robotic systems, especially in dynamic environments. While autonomous vehicle research and development continues at a rapid pace, these systems are becoming more complex and expensive. The objective of this thesis was to determine the feasibility of utilizing a single two-dimensional laser scanning rangefinder for robust obstacle avoidance in unstructured outdoor environments. Specifically, sensing and control algorithms were developed for an autonomous ground vehicle (AGV). The system was designed to operate in varying outdoor environments while avoiding both static and dynamic obstacles. The AGV was able to effectively identify and avoid obstacles within its field of view and to navigate to specific coordinates across variable terrain. While this solution was limited by the sensor used and was not effective in all environments—such as when obstacles encountered were too short to enter the scanner's plane of view—the algorithm developed was successful for visible objects. Small improvements, such as using a gimballed scanner or one that scans in three dimensions, would make this solution more robust for a wider range of environments.

14. SUBJECT TERMS obstacle avoidance, navigation, robot, ROS, outdoor, unstructured, LIDAR, laser, GPS, MATLAB, ground vehicle, autonomous			**15. NUMBER OF PAGES** 105
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

MOBILE ROBOT NAVIGATION AND OBSTACLE AVOIDANCE IN UNSTRUCTURED OUTDOOR ENVIRONMENTS

Calvin S. Hargadine
Lieutenant, United States Navy
B.S., The Citadel, The Military College of South Carolina, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2017

Approved by: Xiaoping Yun
Thesis Advisor

James Calusdian
Co-Advisor

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The ability to detect, characterize, and avoid obstacles is a critical requirement for autonomous robotic systems, especially in dynamic environments. While autonomous vehicle research and development continues at a rapid pace, these systems are becoming more complex and expensive. The objective of this thesis was to determine the feasibility of utilizing a single two-dimensional laser scanning rangefinder for robust obstacle avoidance in unstructured outdoor environments. Specifically, sensing and control algorithms were developed for an autonomous ground vehicle (AGV). The system was designed to operate in varying outdoor environments while avoiding both static and dynamic obstacles. The AGV was able to effectively identify and avoid obstacles within its field of view and to navigate to specific coordinates across variable terrain. While this solution was limited by the sensor used and was not effective in all environments—such as when obstacles encountered were too short to enter the scanner's plane of view—the algorithm developed was successful for visible objects. Small improvements, such as using a gimballed scanner or one that scans in three dimensions, would make this solution more robust for a wider range of environments.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

2D	two-dimensional
3D	three-dimensional
AGV	autonomous ground vehicle
DOF	degrees of freedom
EKF	extended Kalman filter
ENU	East-North-up
GNSS	global navigation satellite system
GNSS/INS	GNSS-aided inertial navigation system
GPL	GNU general public license
GPS	global positioning system
IMU	inertial measurement unit
INS	inertial navigation system
KML	keyhole markup language
LIDAR	light detection and ranging
LRF	laser rangefinder
LTS	long-term support
NED	North-East-down
OS	operating system
OSRF	open source robotics foundation
P3-AT	pioneer 3-AT
PC	personal computer
ROS	robot operating system
RST	robotics system toolbox
SSH	secure shell
USB	universal serial bus
UTM	universal transverse Mercator
WGS84	world geodetic system 1984

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

My advisors, Professor Xiaoping Yun and Dr. James Calusdian, were instrumental in the success of this thesis project. I want to thank Professor Yun for his guidance during the initial phases of research and his assistance with the theory and application of the control algorithms. I also want to thank Dr. Calusdian for his assistance in the lab and his support during the integration and experimentation phases. I deeply appreciate the suggestions and guidance provided by both of my advisors during the writing process. Finally, I would like to thank Professor Brian Bingham for his invaluable help, not only for lending sensors for the project, but also for providing a wealth of knowledge to the Naval Postgraduate School community for autonomous robotic systems.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The research and development of autonomous ground vehicles (AGVs) continues to be a very active area of study, and central to the development of these types of vehicles is the ability to detect and avoid obstacles. Before the AGV can perform its intended function, whether it is moving people or cargo, as with a self-driving car, or performing mapping missions of its surrounding environment, the AGV must be able to navigate to a predetermined location without running into obstacles along the way. The advancement of technology and the desire for more autonomy has resulted in systems that are becoming more complex and more expensive. The ability to perform robust obstacle avoidance using relatively simple sensor suites affords the opportunity to develop these systems at a lower cost. With cheaper vehicles, more can be produced, and cooperative robotic operations can be employed.

The uses for these types of AGVs cover a wide range of functions, from performing land surveys and mapping to delivering mail. In the military realm, these systems can operate in dangerous or denied environments, such as minefields or active combat zones. They can provide services, such as delivering supplies and mobile cover to troops that are pinned down, or covert surveillance in areas that are too risky to employ human forces. The motivation, purpose, and goals of this thesis are outlined in this chapter, as well as previous research conducted in this field.

A. MOTIVATION

AGVs have been the subject of continual research, but as these systems incorporate newer technologies and more advanced guidance algorithms, they become increasingly more expensive. The motivation behind this thesis was the need for an effective navigation and control algorithm for an AGV that minimized complexity and cost. In this thesis, we utilized a simplified sensor suite, along with simple and robust object avoidance and navigation algorithms, to control an AGV. An outdoor, unstructured environment was chosen, as it most appropriately represents the environment in which many AGVs see practical use. Developing a control algorithm to work in this type of

environment allowed for the AGV to operate in more structured environments as well—including indoors. The utilization of a research-grade ground robot chassis provided a flexible platform for rapid development and testing of the control algorithms. Once developed and tested, the sensor suite and associated control software is easily transplanted into more capable or specialized robot frames for future use.

B. PREVIOUS WORK

Object detection, for use in avoidance algorithms, is accomplished using many different types of sensors. Previous work in the field of object detection and avoidance for AGVs include systems using light detection and ranging (LIDAR) sensors, radar, sonar transducers, and image-based systems utilizing monocular or stereo cameras. One of the most common sensors employed for obstacle detection is the LIDAR scanner. AGVs developed using this sensor range from those utilizing a single three-dimensional (3D) LIDAR scanner, as in [1], to more complex systems implementing multiple scanners in different configurations. Shim et al. implemented a high-speed obstacle detection method using five two-dimensional (2D) LIDAR scanners oriented in different planes of view [2]. Other approaches utilized only video—as Cherubini et al. did in [3]—and many fused multiple sensor types together to adequately detect objects within the AGV's operating area, as was the method used in [4], [5], [6], and [7].

Specialized AGVs, such as autonomous cars, use many different kinds of sensors and require very complex control algorithms due to the safety requirements needed to ensure public safety. The sensor suites used on these vehicles, such as the three 3D LIDAR scanners used on a vehicle developed by Shang et al. [8], are expensive and require a significant amount of processing power to operate. Other sensors have been used for this purpose, such as the radar and stereo vision camera used in [9] or the 3D LIDAR, stereo camera, and wheel encoders used in [10], but they also suffered from high cost and large power requirements.

Previous research in obstacle avoidance has provided significant advances in AGV technology and has produced some very complex and capable systems. On the other hand, very simple sensors have been used to control an AGV, such as the sonar-

controlled vehicle in [11], producing systems that were limited by detecting only static obstacles or systems that lose capability when operated outdoors. In this thesis, we attempt to strike a balance between sensor and algorithm complexity and the capability of the autonomous robotic platform.

C. PURPOSE AND GOALS OF THESIS

The purpose of this thesis was to determine the validity of using a single 2D LIDAR scanner for robust obstacle avoidance in dynamic, unstructured outdoor environments. To achieve this purpose, specific goals were developed. The first was the verification and characterization of the laser rangefinder (LRF) and other sensors, such as sonar, to determine if a single sensor had enough resolution and responsiveness for adequate object detection. The next goal was to develop a self-contained autonomous system with enough capability for testing and experimentation. This AGV was required to operate over varying outdoor terrain and in any reasonable weather conditions—excluding severe weather, such as storms or rain. The final goal was to develop robust obstacle avoidance and navigation algorithms that would be effective for both static and dynamic obstacles in unstructured outdoor environments.

In the following six chapters, we discuss the design and implementation of an autonomous robotic system built to achieve the thesis goals. A description of the hardware for the system and the integration of each component is presented in Chapter II. The software suites and the configuration of the underlying communications network are discussed in Chapter III. In Chapter IV, a description of the potential field algorithm is provided, along with details of how the algorithm was implemented for this system. The localization and navigation algorithms are outlined in Chapter V, and the results of experimentation and testing are presented in Chapter VI. Finally, conclusions drawn from the research and testing conducted are presented in Chapter VII, along with recommendations for follow-on work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. HARDWARE DESCRIPTION

The goals developed for this thesis influenced the selection of specific hardware for integration into the robotic system. Since the robot was to operate primarily outdoors in a relatively uncontrolled environment, a four-wheeled chassis was selected that had sufficient driving force to operate on loose surfaces, as well as enough payload and battery capacity to support the other sensors and processing hardware. The 2D LRF was selected based on its rating to operate in outdoor environments, its sufficient angular resolution for detecting small objects commonly encountered outdoors, such as light foliage and branches, and its complementary power requirements for use on the selected robot chassis.

In order to navigate more effectively in an outdoor environment, an inertial measurement unit (IMU) was selected, which incorporated Global Positioning System (GPS) fix information. This sensor provided information used for localization and navigation without relying solely on the chassis wheel encoders, which are prone to error accumulation. To provide system autonomy, a miniature personal computer (PC) was chosen as the interface between all of the sensors and the robot chassis. This PC was powered solely from the robot's onboard batteries and provided the computing power needed to run the required software and navigation algorithms. The details of each hardware component are discussed, as well as how they were integrated together into the autonomous robotic system.

A. OMRON ADEPT MOBILEROBOTS PIONEER 3-AT

The Pioneer 3-AT (P3-AT) mobile robot, developed by Omron Adept MobileRobots, LLC, is a four-wheeled outdoor robotic development platform. According to the MobileRobots website and associated datasheet, respectively, the P3-AT "is a highly versatile four-wheel drive robotic platform" [12, p. 1] that provides mobility over a range of surfaces including "asphalt, flooring, sand, and dirt" [13, p. 1]. The aluminum wheels with 21.5-cm pneumatic tires provide 7.0 cm of ground clearance, and the chassis is rated for up to 20 kg of payload capacity [14]. The wheels are driven by four reversible

direct-current motors in a skid-steer drive configuration providing a zero turn radius and a 34-cm swing radius [15]. Each motor is "equipped with a high-resolution optical quadrature shaft encoder for precise position and speed sensing and advanced dead-reckoning" [15, p. 11]. These encoders provide 34,000 counts per revolution for use in estimating the position of the P3-AT as it moves [15]. The base configuration of the P3-AT, including optional sonar sensors, bumper switches, and emergency stop button, is illustrated in Figure 1.

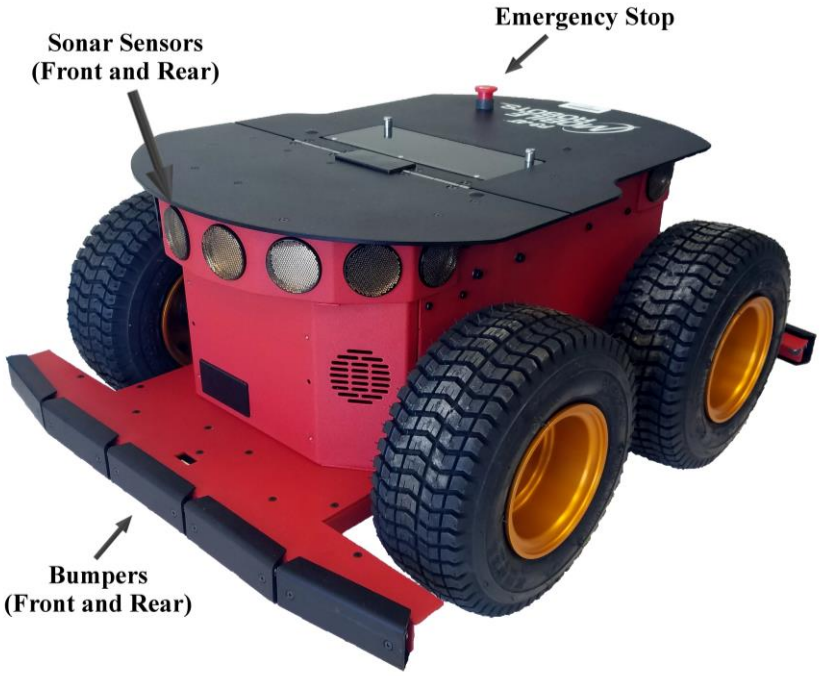

Figure 1. Pioneer 3-AT Mobile Robot Base Configuration

The integrated sonar sensors provide 360-degree sonar coverage utilizing eight transducers each on the front and rear of the chassis. Six transducers are angled in 20-degree increments, and the remaining two are positioned on each side of the chassis to provide sonar ranges to obstacles between the minimum effective range of 10 cm and its maximum range of 5.0 m. The transducers are fired in sequential order within each bank, one every 40 ms, for a complete collection of sonar ranging data every 320 ms [15]. Ten bumper switches, five in each bank, front and rear, provide sensing for the detection of

obstacles that come into contact with the robot chassis, and an emergency stop push button immediately stops all drive motors for the robot when depressed.

Up to three hot-swappable lead-acid batteries, for a combined rating of 27 Ah at 12 VDC, provide power for the P3-AT. To drive additional sensors or accessories, the P3-AT provides regulated 5.0 VDC at 1.5 A and battery 12 VDC at 2.0 A connections via a Motor-Power Distribution Board [15]. An onboard Renesas SH2-based 32-bit microcontroller serves as the interface between the chassis components and provides status indication, communication, and input controls to the user via a User Control Panel [15]. This panel includes a serial RS-232 connection, light-emitting diodes to indicate the status of power, battery, transmit, and receive operations, and push buttons to reset the microcontroller as well as control motor and auxiliary power, as illustrated in Figure 2.

Figure 2. P3-AT User Control Panel

B. HOKUYO AUTOMATIC UTM-30LX

The Hokuyo UTM-30LX 2D scanning LRF is a compact LIDAR sensor utilizing emitted laser energy to provide highly accurate range data with high angular resolution. This particular LRF was developed for use not only in an indoor environment as with

many LIDAR systems of this class but also for use outdoors. Hokuyo describes the UTM-30LX on its product information webpage as "suitable for robots with higher moving speed because of the longer range and fast response" of this LRF [16, p. 1]. The UTM-30LX is powered by the P3-AT's batteries, operating nominally at 12 VDC and is shown in Figure 3. The ranging information, along with control signals and status information, is transmitted to and from the sensor via a Universal Serial Bus (USB) version 2.0 connection [17].

Figure 3. Hokuyo UTM-30LX Laser Scanning Rangefinder. Source: [16].

The LRF utilizes a class 1 semiconductor laser with an operating wavelength of 905 nm. A DC motor rotates a mirror at a speed of 25 ms per rotation to scan the laser in a 2D plane parallel to the mounting plane. The LRF has a field of detection of 270 degrees with 1,080 data points per scan, resulting in an angular resolution of 0.25 degrees [17]. The LRF is able to detect objects and report their ranges from anywhere within its field of view and effective range—from 0.1 m to 30 m [16]. The LRF has a relatively small form factor, at 60 mm width, 60 mm depth, and 87 mm height, and weighs only

210 g without cables. These characteristics along with its low power requirements—less than 8.0 W—make this scanner ideal for autonomous robotic applications [17].

C. LORD MICROSTRAIN 3DM-GX5-45

The 3DM-GX5-45 inertial navigation system (INS), developed by LORD MicroStrain and depicted in Figure 4, is an IMU with nine degrees of freedom (DOF). This particular IMU also incorporates Global Navigation Satellite System (GNSS) inputs obtained through a separate GNSS antenna connected via a micro-miniature coaxial connection to the IMU body. As stated on the 3DM-GX5-45 datasheet, this INS is an "all-in-one navigation solution" featuring a "high-performance, integrated, multi-constellation GNSS receiver" along with "fully calibrated, temperature-compensated, and mathematically-aligned" sensor measurements [18, p. 1]. The sensors used for this device incorporate Micro-Elecro-Mechanical System technology and provide "a highly accurate, small, light-weight device" [18, p. 1].

Figure 4. MicroStrain 3DM-GX5-45 GNSS-Aided Inertial Navigation System. Source: [19].

The sensors included in the IMU portion of the GNSS-aided inertial navigation system (GNSS/INS) include three gyroscopes, three accelerometers, and three magnetometers, providing nine DOF. These sensors provide outputs in an orthogonal coordinate system, discussed in more detail in Chapter V, and provide automatic magnetometer calibration and anomaly rejection, as well as compensation for vehicle

noise and vibrations [19]. The integrated GNSS receiver utilizes not only GPS but also the GLONASS, BieDou, and Galileo GNSS constellations.

In addition to the traditional IMU outputs of roll, pitch, yaw, and heading, the GNSS/INS provides GNSS outputs, raw sensor outputs for acceleration, angular rate, and ambient pressure, as well as filtered position, velocity, and altitude estimates via an onboard Auto-Adaptive Extended Kalman Filter (EKF). Dual on-board processors run this filter using data from the IMU sensors and GNSS inputs, and the system provides status messages for both the EKF estimation and GNSS fix information [18]. Communication and power for the GNSS/INS is provided via USB version 2.0.

D. SLIMPRO SP675P MINI PC

The SlimPRO SP675P is a miniature PC that runs off the P3-AT battery supply, nominally at 12 VDC. The SlimPRO's small form factor (42 mm height, 146 mm width, and 254 mm length) and low weight (2.4 kg) make it a good choice for robotic applications [20]. The SlimPRO has many input and output connections including Video Graphics Array, RS-232 nine-pin serial, Ethernet, and six USB ports (four version 3.0 and two version 2.0), as illustrated in Figure 5. The SlimPRO has a built-in Wi-Fi adapter for communications that was used to provide Secure Shell (SSH) connections between the SlimPRO and a remote laptop for startup, diagnostics, and shutdown of the robotic system during testing.

The SlimPRO used for this thesis research included a 64-bit dual-core Intel Pentium Central Processing Unit, model B950, running at 2.1 GHz, 16 GB of double data rate type three small-outline dual in-line memory modules, and integrated Intel Sandy Bridge Mobile graphics. A 300 GB internal hard drive provided ample space for file storage during testing and development. With these features, the SlimPRO PC provided sufficient computing power for the algorithms and sensors utilized in the robotic system.

Figure 5. SlimPRO Mini PC Input/Output Connections. Source: [20].

E. INTEGRATED CHASSIS CONFIGURATION

Individual hardware components were tested separately (testing is discussed in more detail in Chapter VI) and then integrated with the P3-AT chassis via a common aluminum frame. The frame was designed and built to allow the LRF an unobstructed view and to provide enough surface area for mounting the sensor hardware and SlimPRO PC with some extra room for expansion if additional hardware was desired. The final configuration of the robotic system is illustrated in Figure 6. The mounting position of the LRF was chosen to allow for the detection of the majority of obstacle types that the robot would encounter in an outdoor environment. The mounting height of the LRF was low enough to detect shorter obstacles but high enough to minimize false detections of the ground when the robot was traversing rough terrain.

Figure 6. Integrated Robotic System Hardware

A large aluminum heat-sink plate was mounted to the base of the LRF with thermal compound to prevent overheating during operation. The SlimPRO PC was installed on the upper portion of the frame at the rear to maximize its distance from the GNSS/INS and provide unobstructed access to the P3-AT User Control Panel and emergency stop button. The GNSS/IMU was attached centrally on the upper portion of the frame to minimize the offset between its reference frame and the robot reference frame but as far away from the P3-AT drive motors and the SlimPRO as possible to minimize electromagnetic interference during operation. The GPS antenna was magnetically mounted to the top of the SlimPRO to provide an unobstructed view of GNSS constellations. Finally, a Microsoft LifeCam HD-3000 web camera was mounted near the GNSS/INS to provide a visualization of the environment as the system navigated itself to its goal. The LRF and SlimPRO PC were powered directly from the P3-AT's battery bank via the Motor-Power Distribution Board, nominally at 12 VDC. All other

sensors were powered via USB connection to the SlimPRO, which also provided communications to and from the sensors. The SlimPRO communicated with the P3-AT microcontroller via a nine-pin RS-232 serial connection.

Once the system hardware was integrated into the P3-AT chassis, work began on the underlying software architecture used for the command and control of the system. The software suites and specific configuration used is described in Chapter III.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SOFTWARE DESCRIPTION

With the hardware components chosen, software suites were selected that would most effectively achieve the thesis research goals. These suites had to provide for the integration of all sensors as well as provide an environment for continued development and testing. The Robot Operating System (ROS) was chosen as the underlying command and control network for the system because of its versatility, interoperability, and modularity. The modularity of ROS allows multiple programming languages to be used in the development of the system, which facilitated rapid development and continual modification. MATLAB was chosen to build and implement the obstacle avoidance and navigation algorithms as it includes a toolbox for interfacing with ROS and provides a common workspace for modification and iteration during testing. In order to safely implement and test the obstacle avoidance algorithm, a simulation program was required. Gazebo was chosen for simulation because of its integration with ROS and its robust feature set. The built-in ROS visualization tool, rviz, was used during sensor testing and verification because of its ease of use and its built-in support.

The SlimPRO, which ran the ROS network and MATLAB, utilized the Ubuntu Linux 14.04 long-term support (LTS) operating system (OS). As the ROS architecture was designed for use on Linux, specifically Ubuntu Linux, this OS was chosen. The external laptop on which the Gazebo simulation, some of the algorithm development, and initial sensor testing was accomplished was dual-booted with Ubuntu Linux 16.04 LTS and the Windows 7 OS. A newer version of Ubuntu Linux was chosen for the laptop to facilitate the use of an updated version of Gazebo that would not run on the older 14.04 LTS. The laptop also had Windows installed for initial sensor testing, discussed in Chapter VI, as many tools used for testing were designed for Windows.

A. ROBOT OPERATING SYSTEM

ROS is an open-source collaborative product developed by the Open Source Robotics Foundation (OSRF). According to OSRF, “ROS was built from the ground up to encourage *collaborative* robotics software development” and “is a collection of tools,

libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide body of robotic platforms" [21, p. 1]. The architecture of ROS allows for the control of complex robotic systems across multiple platforms and networks. Each ROS network consists of a master and one or more nodes and topics. A ROS network is modular and dynamic in that each node runs independently of the other nodes in the system and can be started and/or stopped without affecting the other nodes running in the network. ROS accomplishes this by enabling communication between the nodes via messages published on specific topics. This system uses a publisher and subscriber architecture in that when a node needs to provide information to the network, it publishes the information in the form of a message on a pre-defined topic. When a node needs to pull information from the network, it subscribes to a specific topic and is able to receive the messages that are published to that topic. In order to establish all of the connections within this dynamic system, a ROS master is used. The ROS master is started before any other nodes on the network, and when each subsequent node is started, it registers with the master node. As part of its registration, each node also provides the master with its publication and subscription information. The master uses this registration information to keep track of all of the nodes, messages, and topics so that when a new node registers on the network that interfaces with an existing part of the network, the master can update the affected nodes with new connection information. Once the registration process is completed, the nodes exchange information directly between each other over established topics.

A simple example network consisting of a ROS master and three nodes is shown in Figure 7. In this example network, the three nodes of the system register with the master, illustrated as dashed lines, and communicate over two topics. Node 1 publishes to Topic 1, which has two subscribers—Nodes 2 and 3. Node 3 publishes to Topic 2, which only has Node 1 as a subscriber. This network also includes another construct of ROS—service servers and clients. The service server, which can be a stand-alone node or included as part of another node, provides a service for incoming client requests. An example of a simple service is an addition service, where the requesting client provides two numbers as arguments to their service request, and the service server adds the two

arguments together before providing the requesting node with the result. In the example network, all nodes have access to the service server and can make service requests independent of their communications with the other nodes in the system.

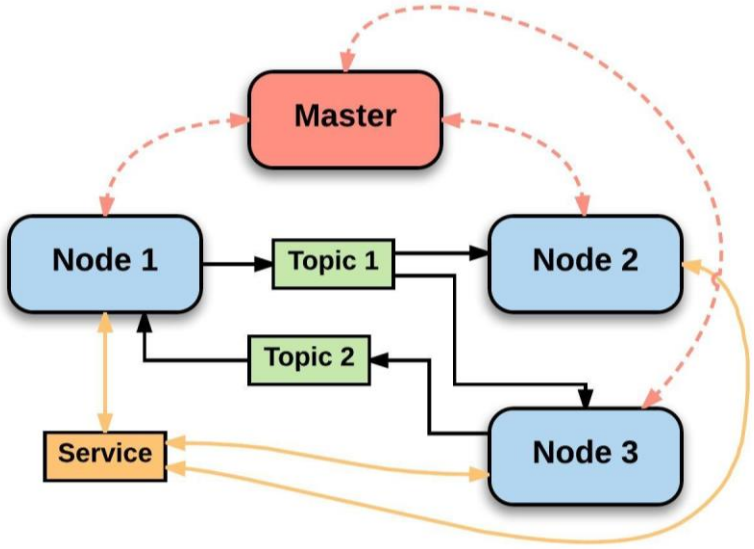

Figure 7. Example ROS Communications Network

The design of the ROS network for this thesis research was based on providing a node for each sensor element as well as one to control the P3-AT chassis. In addition to those nodes that were required for the hardware interfaces, additional nodes were developed for MATLAB, which executed all of the motion planning algorithms and decision logic as well as performing the necessary coordinate system transformations from a latitude and longitude frame to a local X and Y frame. A diagram of the nodes and major topics used for this thesis research is shown in Figure 8. Although this diagram does not include all of the elements of the ROS network used, such as the ROS master, service servers, and diagnostic information, it does show the major components required for operation of the developed robotic system.

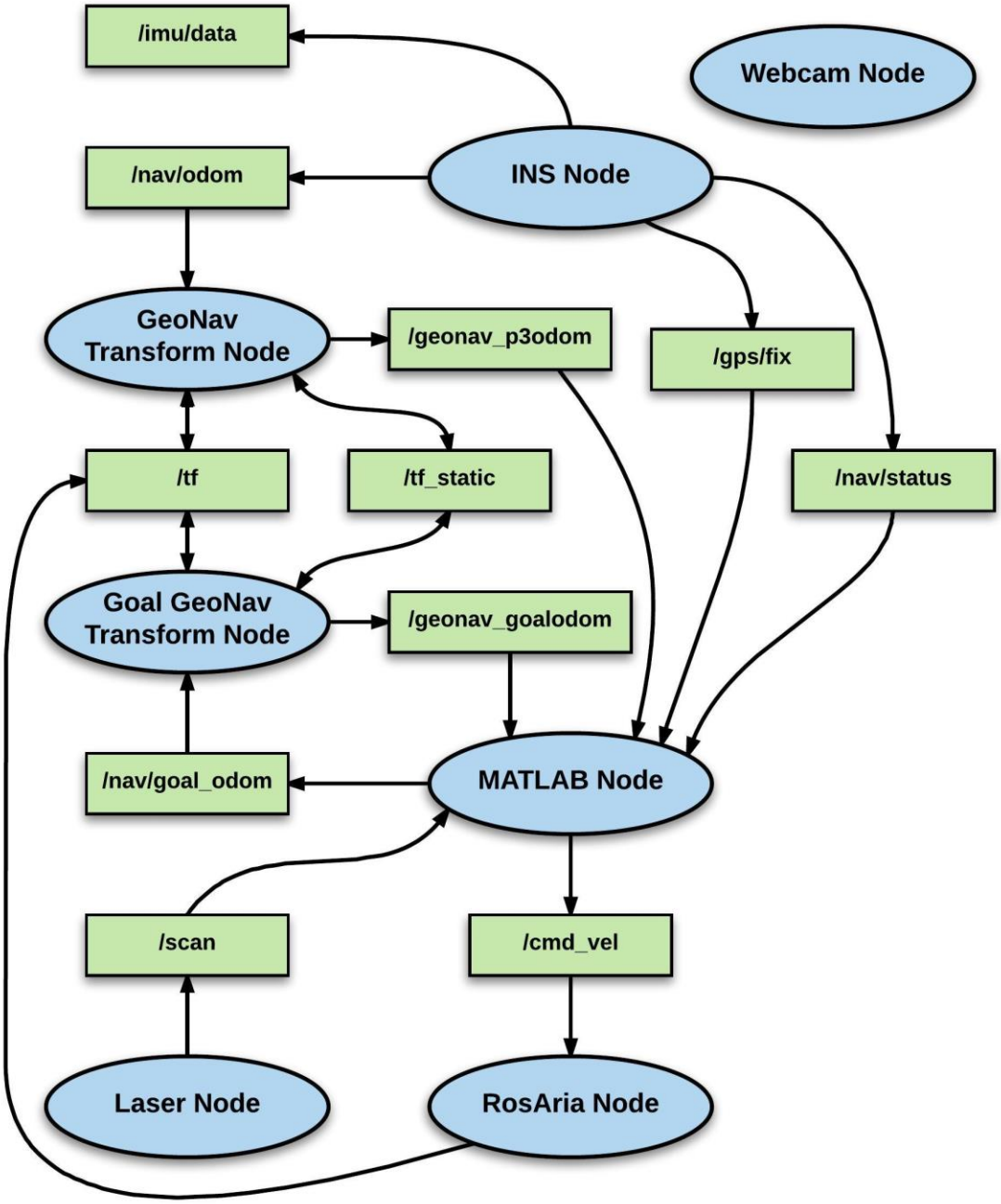

Figure 8. ROS Network Nodes (Blue) and Topics (Green)

The node used to interface with the LRF was called the hokuyo_node. This node, authored by Brian P. Gerkey, Jeremy Leibs, and Blaise Gassend, provided configuration and communication between the ROS network and the Hokuyo LRF and

was used under the GNU Lesser General Public License [22]. The hokuyo_node, renamed Laser Node for this thesis, allowed for the configuration of specific laser parameters such as the minimum and maximum scan angles, the communications port in use on the SlimPRO, and setting the intensity mode of the laser. Once running, the Laser Node published laser scan data including laser ranges, minimum and maximum scan angles, and angular resolution, on the /scan topic for use by the MATLAB Node.

The GNSS/INS was controlled by the microstrain_3dm_gx5_45 node, renamed INS Node for this thesis research. This package, used under the GNU General Public License (GPL), was authored by Brian Bingham and included another node used in this thesis research—the geonav_transform node [23]. The INS Node not only provided configuration for the GNSS/IMU but also provided information published on four different topics for use by the ROS network. The raw IMU data was published on the /imu/data topic for diagnostic use, and the raw GPS fix information was published on the /gps/fix topic. The output of the onboard EKF was published on the /nav/odom topic, and the status of the EKF was published on the /nav/status topic. To facilitate the necessary coordinate transformations, discussed in more detail in Chapter V, two instances of the geonav_transform node were implemented, renamed the GeoNav Transform Node and the Goal GeoNav Transform Node. Each of these nodes performed a coordinate transformation from the World Geodetic System 1984 (WGS84) latitude and longitude coordinates to the corresponding Universal Transverse Mercator (UTM) and local X and Y coordinates based on an initial datum parameter. The difference between these two nodes is that the GeoNav Transform Node performs the transformation using the robot's current fix information whereas the Goal GeoNav Transform Node only transforms the goal position. The outputs of these nodes, published on the /geonav_p3odom topic for the robot position and the /geonav_goalodom topic for the goal position, provide a common reference frame for use in the MATLAB navigation algorithm.

The onboard microcontroller for the P3-AT directly controlled the motors, bumper switches, and sonar transducers on the robot chassis, and a ROS node was required to interface with the controller to send it commands and publish resultant robot

data back to the ROS network. The ROSARIA package, used under the GPL and authored by Srećko Jurić-Kavelj, included a RosAria node to communicate with the onboard microcontroller using Omron Adept MobileRobot's open source Advanced Robot Interface for Applications library [24]. This library was used by the P3-AT microcontroller to control the "robot's velocity, heading, relative heading, and other motion parameters" dynamically and also received "position estimates, sonar readings, and all other current operating data sent by the robot platform" [25, p. 1]. The RosAria Node, as named for this thesis research, subscribed to the /cmd_vel topic published by the MATLAB Node, and published coordinate frame transformation information on the /tf topic. The RosAria Node commanded the linear and angular velocities for the robot as received on the /cmd_vel topic. It also provided other robot diagnostic information back to the ROS network, such as battery voltage, bumper switch states, sonar ranges, and robot pose information based on the onboard motor encoders.

As an additional way of visualizing the operating environment of the robot and to correlate the laser scan data with physical objects, a web camera was installed on the robot. The node used to interface with this camera was the usb_cam_node, used under the Berkeley Software Distribution license and authored by Benjamin Pitzer [26]. This node, renamed Webcam Node for this thesis, was part of the usb_cam ROS package and published image data from the camera to the ROS network. The robot did not use the image data in any capacity for object avoidance or navigation—it served only to visualize what the robot was encountering as it maneuvered through its environment.

To simplify the startup of the ROS network when the robot was powered on, a launch file was generated to start up each individual node with its associated parameters. The launch file, provided in Appendix B, when executed started the ROS master and then each ROS node sequentially. The status of each node, including debugging and diagnostic information, was printed in the terminal window to ensure that each was started and running correctly. Once all of the other ROS system nodes were running, the MATLAB Node and the robot control algorithms, discussed in Chapters IV and V, were started.

B. MATLAB

MATLAB, developed by The MathWorks, Inc., is a development environment utilizing a matrix-based programming language [27]. The software is stable, robust, and under continuing development. Aside from the standard MATLAB environment, it also includes numerous toolboxes that add functionality for the developer in specific areas. The Robotics System Toolbox (RST) is a robot development toolbox for MATLAB that provides the ability to "create ROS nodes in MATLAB and Simulink, exchange messages with other nodes on the ROS network, import ROS log files into MATLAB, and generate C++ code for a standalone ROS node" [28, p. 1]. The RST provides an interface between the development environment of MATLAB and the ROS network that allows a user to "communicate with a ROS network, interactively explore robot capabilities, and visualize sensor data" [28, p. 1]. The MATLAB environment was chosen for the development and execution of the obstacle avoidance and navigation algorithms because of its ease of use, the ability to debug and step through the program in real time, and its compatibility with ROS provided by the RST.

The RST in MATLAB communicated with the ROS network by registering with the master in the same way as any other node. Once the ROS network was running, the RST registered a MATLAB Node with the master. The MATLAB Node subscribed to the /gps/fix, /nav/status, /scan, /geonav_p3odom, and /geonav_goalodom topics and published to the /nav/goal_odom and /cmd_vel topics. The subscriber callback function in MATLAB continually monitored the specified topic, and each time the message on the topic was updated, it updated an associated global MATLAB variable. The subscriber callback process ran in the background, so other programs could be executed at the same time. This process was repeated for each topic for which MATLAB was a subscriber, and the resultant global variables in MATLAB were representative of the current state of the robotic system. These variables were then used in the obstacle avoidance and navigation algorithms to determine the desired linear and angular velocities of the robot, as discussed in more detail in Chapters IV and V, and then published on the /cmd_vel topic for use by the RosAria Node.

To facilitate the user entering desired goal coordinates, the MATLAB script read in a text file containing the goal and provided the coordinates to the ROS network via publication on the /nav/goal_odom topic. This data was used by the Goal GeoNav Transform Node to transform the goal coordinates from WGS84 to the local X and Y coordinate frame. To publish the goal odometry and the robot velocities, each message was built as a variable in MATLAB and then pushed to the publisher function, provided by RST, for publication on the ROS network.

C. GAZEBO SIMULATOR AND RVIZ

Simulation of the basic chassis and function of the robotic system allowed for the rapid development and testing of the obstacle avoidance algorithm. Requirements for the simulation program included the ability to integrate well with the ROS network, have support for the sensors used, specifically the LRF, and provide the ability to add or remove objects while the simulation was in progress in order to simulate dynamic obstacles. The simulator chosen, based on these requirements, was Gazebo. Dr. Andrew Howard and Nate Koenig originally developed this 3D simulator in 2002 at the University of Southern California. Since then it has been continually improved, and development of the simulator was taken over by the OSRF in 2012 [29]. According to the overview of Gazebo on its webpage, "Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments," and offers "physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs" [30, p. 1]. This simulator includes a large library of robot models, supports many different sensors via plugins, and since it was developed alongside ROS, provides a ROS package called gazebo_ros_pkgs to facilitate communications between the simulator and the ROS network [31]. This package, authored by John Hsu, Nate Koenig, and Dave Coleman, is a wrapper for the standalone Gazebo program that provides an interface with ROS using "ROS messages, services, and dynamic reconfigure" [32, p. 1].

The Gazebo simulator provides user interfaces for designing worlds and robot models for use in simulation. For this thesis research, a robot model was developed

utilizing two existing open-source models from the Gazebo model library—one for the P3-AT chassis, authored by Dereck Wonnacott [33], and one for the Hokuyo LRF, authored by John Hsu [34]. These two models were combined using the model editor, and plugins for the robot's skid-steer drive and the laser data from the LRF were added. Since the only sensor required to implement the obstacle avoidance algorithm was the LRF, no additional sensors were modeled for simulation. The resultant robot model used for simulation is shown in Figure 9.

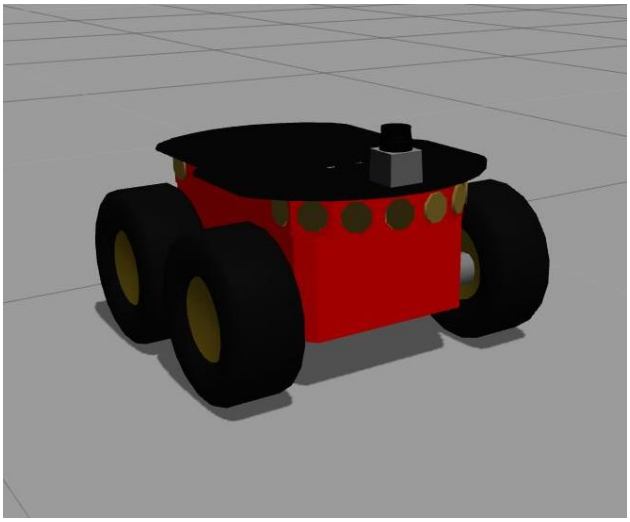

Figure 9. Gazebo Model of P3-AT with LRF Attached

The world developed for simulation was barren except for four spherical objects used to test the laser and robot functionality. As algorithm development and testing progressed, the details of which are discussed in Chapter V, more obstacles of varying shapes and sizes were added to the world, testing different aspects of the obstacle avoidance algorithm. An example of the initial simulation environment, including the robot model and initial obstacles, is illustrated in Figure 10. The robot model was initialized at the origin, and the blue rays visible in the figure represent the LRF ranges, including returns from the visible objects.

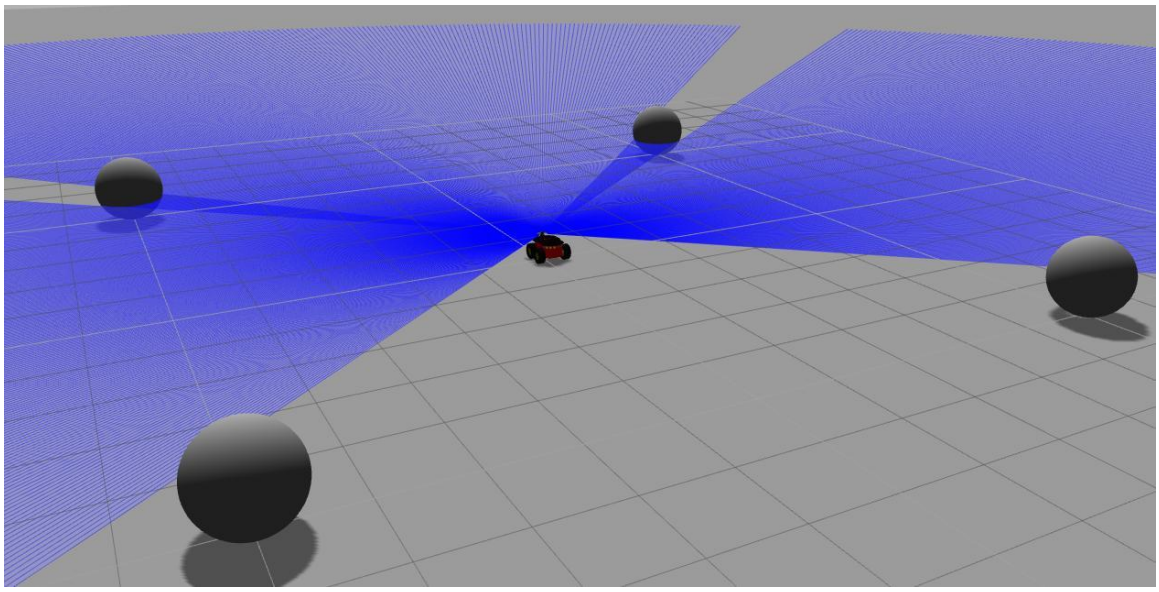

Figure 10. Initial Gazebo Simulation Environment

While Gazebo was used to simulate the robotic system in order to develop the object avoidance algorithm, rviz was used prior to system integration to test and verify the proper operation of individual sensors. Morgan Quigley, Brian Gerkey, and William D. Smart, in their book *Programming Robots with ROS*, describe rviz as a general-purpose 3D visualization environment used for robots and sensors [35]. This tool, included in the standard ROS package, provides a method for visualizing nearly all aspects of the data transmitted on the ROS network. For this thesis research, rviz was used to visualize laser, sonar, and webcam data to verify sensor operation and to visualize the robot's environment. An example rviz interface showing laser scan data is shown in Figure 11. White dots in the figure represent returns from the LRF, and the robot's environment is visualized as these returns form the outlines of obstacles. The returns are shown from a top-down view with the LRF centered at the origin. The visualization of the data provided by rviz was used to make both quantitative and qualitative assessments of sensor effectiveness during sensor testing, and when used in conjunction with webcam images, was used to correlate objects in the robot's field of view.

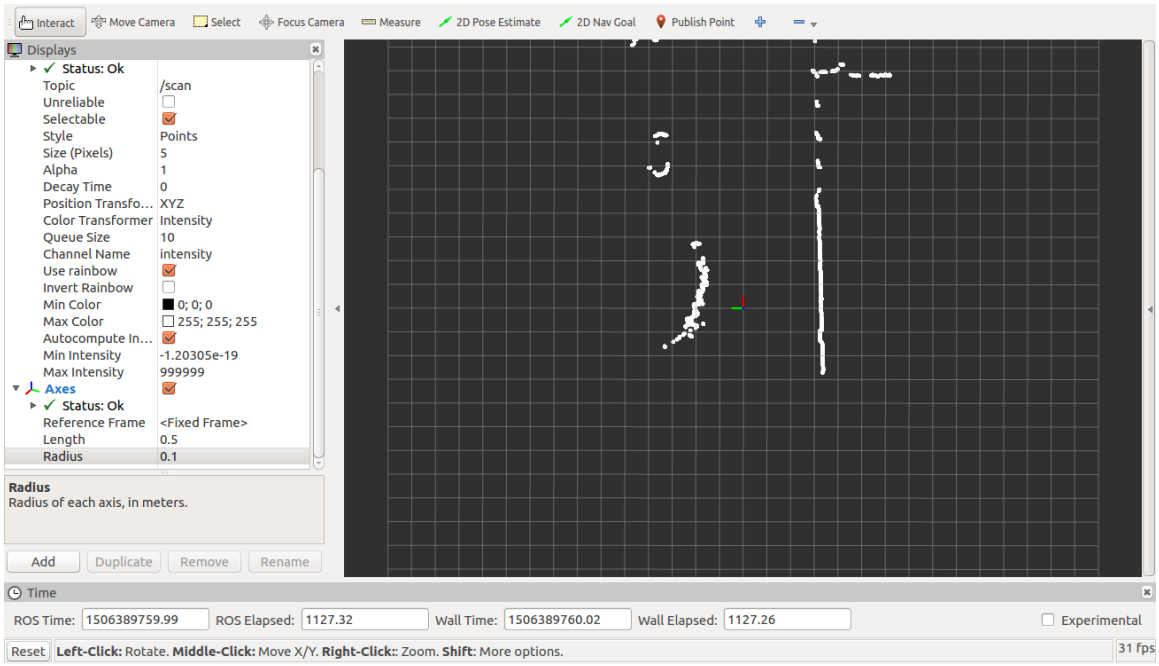

Figure 11. Example rviz Interface Displaying LRF Data

The software suites described in this chapter provided a robust environment for the development and testing of the robotic system. Once the software framework was established, work on the creation and refinement of the AGV's controlling algorithms commenced. A detailed description of the obstacle avoidance, localization, and control algorithms are provided in Chapters IV and V.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. OBSTACLE AVOIDANCE ALGORITHM

Obstacle avoidance is a fundamental problem for any autonomous system as it attempts to reach its destination. The goals of this thesis research require the robot to be able to detect and avoid both static and dynamic obstacles. With this in mind, the system was developed based on the most limiting condition—dynamic obstacles—and was designed for obstacles with speeds up to a human walking pace. The selection of the LRF as the sensor of choice for obstacle detection, discussed in detail in Chapter II, allowed for the detection of objects in high resolution, including those objects less than an inch wide. The object data was then used in the potential field algorithm to control the behavior of the AGV. The potential field algorithm is discussed in this chapter along with a discussion of the approach used to deal with one of the major drawbacks of this algorithm—the local minimum problem.

A. POTENTIAL FIELD ALGORITHM

Many different algorithms have been developed to facilitate obstacle avoidance for autonomous systems. Many of these techniques are based on having a prior knowledge of the operating environment. These methods use the known positions of the obstacles in the environment to build a path for use by the robot. Cell decomposition, as described by Jean-Claude Latombe in *Robot Motion Planning* [36], is a method in which the environment is divided into non-overlapping cells, and then a connectivity graph is developed to represent the adjacency of these cells. Next, a path for the robot is created by using a channel connecting the cell with the robot's starting position and the cell containing the goal [36]. The cell decomposition method can provide an efficient path to the goal for the robot, but the requirement of complete knowledge of the environment limits its use. This technique does not work well in outdoor environments, where many variables can change, and the prior knowledge requirement restricts the available environments to those with only static obstacles.

To achieve the goals of this thesis research, a technique that does not rely on prior knowledge of the environment was required. The bug algorithm and its variants, as

described by Choset et al. in *Principles of Robot Motion: Theory, Algorithms, and Implementations*, are straightforward techniques requiring minimal sensor suites, and the success of the algorithm is guaranteed when a path to the goal is possible [37]. In the bug algorithm, the robot starts and heads directly toward the goal position. Once the robot encounters an obstacle, it turns and follows the obstacle until it reaches a point where progress toward the goal is possible. The robot once again heads toward the goal, and this process is repeated until the robot reaches its destination. This algorithm is simple, but it can be inefficient based on the variant used and the configuration of the obstacles present. The bug algorithm also assumes the robot has perfect localization information [37] and can break down when the environment is dynamic.

The obstacle avoidance technique chosen for this thesis, based on its goals, was an artificial potential field algorithm. In this type of algorithm, as described by Latombe in [36], the robot is treated "as a particle under the influence of an artificial potential field" [36, p. 295]. Latombe continues by stating that the total artificial potential field is characterized "as the sum of an attractive potential pulling the robot toward the goal…and a repulsive potential pushing the robot away from the obstacles" [36, p. 295]. Another way of visualizing how the artificial potential field affects the robot, as presented in [37], is to think of the robot as a positively charged particle being attracted to a negatively charged goal. In this case, the obstacles present in the environment also have a positive charge, providing a repulsive force on the robot and forcing it away from the obstacles [37]. This type of algorithm provides both obstacle avoidance and navigation for the robot, as the attractive potential drives it toward its goal while avoiding the repulsive potentials of the obstacles.

The total potential field U is a combination of an attractive potential field U_{att} and a repulsive potential field U_{rep}

$$U = U_{att} + U_{rep}. \tag{1}$$

This total potential field is used to generate a force vector $\vec{F}(q)$, which is utilized to control the robot at a point q. This force vector is defined as

$$\vec{F}(q) = -\vec{\nabla}U(q), \tag{2}$$

where $\vec{\nabla}U(q)$ is the gradient vector of the total potential field U at point q [36]. By using the negative gradient of the potential field, the robot is forced to move toward the lowest potential present in the total field. An advantage of the artificial potential field is that regardless of where the robot starts, it moves toward the goal from an area of high potential to an area of low potential. The goal is located at the point of lowest potential within the environment, so the robot automatically stops once it reaches this goal. A byproduct of introducing an additive repulsive potential within this field is that it creates the possibility for local minima of potential in the total field at locations other than the goal. If the robot is solely influenced by the total artificial potential field when it encounters one of these local minima, it stops. This is a disadvantage of using this type of algorithm, and requires additional control logic to provide an escape mechanism for the robot if it encounters a local minimum.

The forces acting on the robot, due to the artificial potential field, were re-calculated continuously as the robot moved through its environment. By continually re-evaluating its surroundings, the potential field algorithm allowed the robot to reach its destination even as obstacles changed around it. In the rest of this chapter, we discuss the development of the artificial potential field, including its associated attractive and repulsive components, as well as the technique used to provide an escape mechanism for the robot when it is trapped in a local minimum.

B. ATTRACTIVE FORCE CALCULATION

The first component developed for the total potential field U was the attractive potential. This potential field U_{att} at point q, as defined by Latombe in [36], is

$$U_{att}(q) = \frac{1}{2}\xi\rho_{goal}^2(q), \tag{3}$$

where ξ is a positive scaling factor and $\rho_{goal}(q)$ is the Euclidean distance between point q and the goal position. This definition of the attractive potential creates a parabolic well

with its minimum at the goal position. An attractive force vector $\vec{F}_{att}(q)$ at point q was developed as the negative gradient of this field

$$\vec{F}_{att}(q) = -\vec{\nabla} U_{att}(q) \tag{4}$$

and is used in conjunction with the repulsive force vector to control the movement of the robot [36]. An example attractive potential field is shown in Figure 12. The goal, highlighted in white, is located at an X-position of two meters and a Y-position of negative four meters. The resultant force on a robot in this field is analogous to a ball starting from any position within the field and rolling down to the minimum at the goal due to gravity.

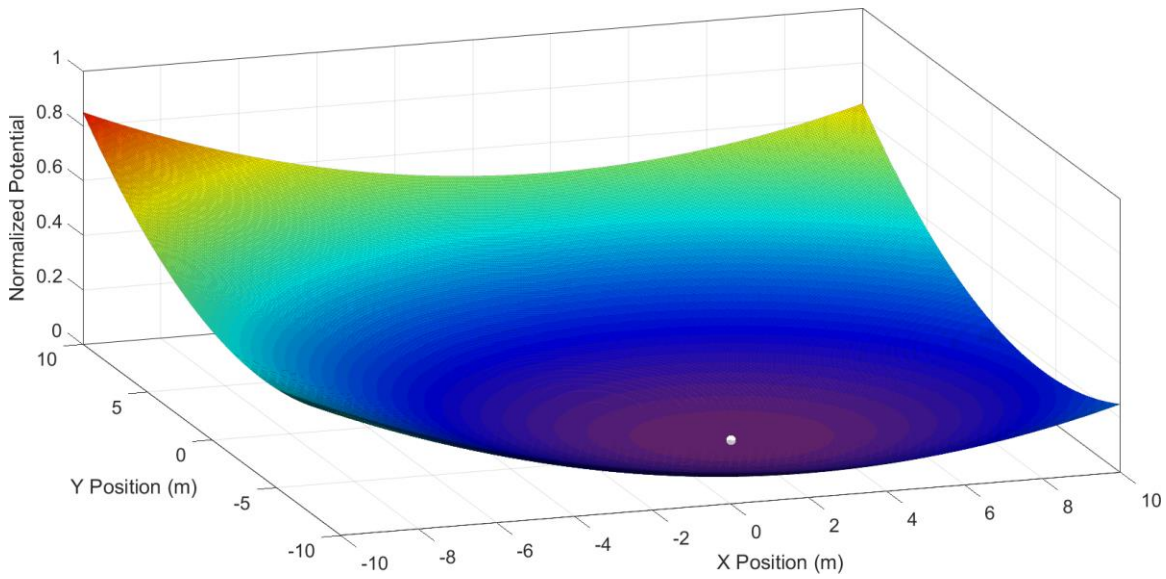

Figure 12. Example Attractive Potential Field with Goal at Coordinates (2, −4)

The attractive force vector used for this thesis research was developed using this method. The magnitude of the vector was calculated using the distance from the current robot position and the goal coordinates. Without a limit, the magnitude of the resultant attractive force grows without bound as the distance from the goal increases. A parameter *d* was used in the algorithm to define a distance from the goal at which a transition occurred in the magnitude of the attractive force. When the robot's distance to the goal was less than or equal to *d*, the magnitude of the attractive force was calculated based on

a parabolic potential well described by Equation (3). When the robot's distance from the goal was greater than *d*, a maximum value was assigned to the attractive force corresponding to the maximum safe translational velocity of the robot. The direction of the attractive force vector was calculated as the angle error between the robot's current heading and the heading required to reach the goal. This attractive force vector was combined with a repulsive force vector, described in the next section, to generate a total force vector. The total force vector was then used to generate translational and rotational velocities for robot motion.

C. REPULSIVE FORCE CALCULATION

The repulsive potential field was the component of the total potential that facilitated obstacle avoidance. According to Latombe, the goal of the "repulsive potential is to create a potential barrier" around all obstacles "that cannot be traversed by the robot" [36, p. 299]. To prevent the repulsive potential from affecting the attractive potential when the robot was sufficiently far enough away from obstacles, a "distance of influence" [36, p. 300] parameter ρ_0 was used to define when the repulsive potential field was forced to zero. The repulsive potential field $U_{rep}(q)$, as defined by Latombe, at point *q* is

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(q) \le \rho_0, \\ 0 & \text{if } \rho(q) > \rho_0, \end{cases} \tag{5}$$

where η is a positive scaling factor and $\rho(q)$ is the distance from the point *q* to an obstacle [36]. An example repulsive potential field is illustrated in Figure 13. The function from Equation (5) was used to generate the figure with three obstacles present within the space. The magnitude of the repulsive potential tends toward infinity as the distance to the obstacle boundary goes to zero. Moving away from the obstacle, the repulsive potential decays to zero as the distance from the obstacle approaches and exceeds ρ_0.

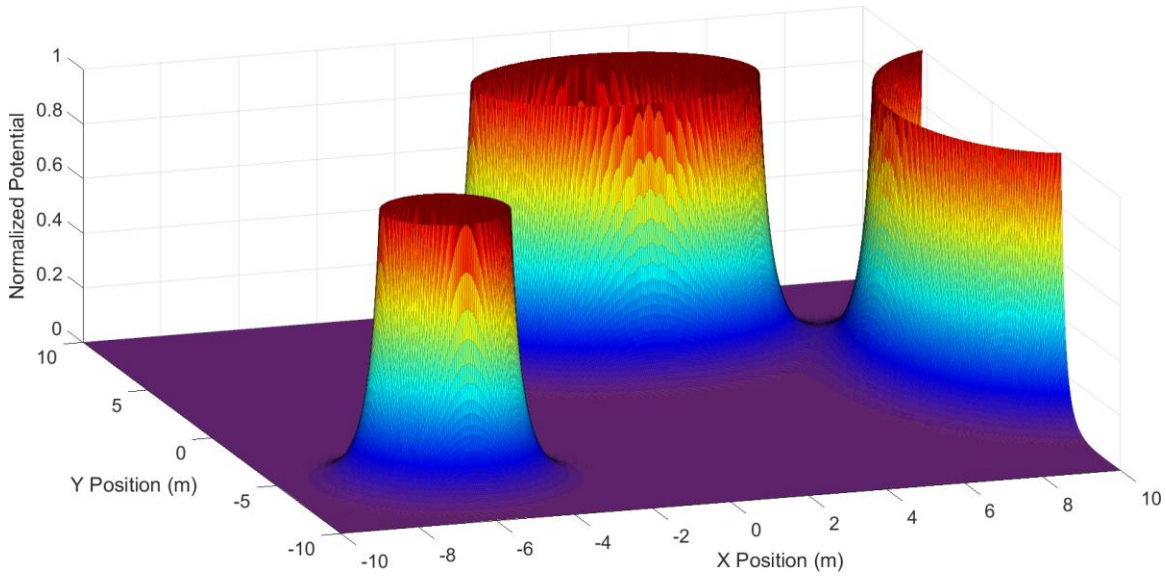

Figure 13. Example Repulsive Potential Field

These two potential fields are combined using Equation (1) to produce a total potential field to facilitate obstacle avoidance and navigation. An example total potential field is illustrated in Figure 14, as a summation of the data present in Figures 12 and 13. This total field is representative of the types of potentials encountered by the AGV as it traverses its environment.

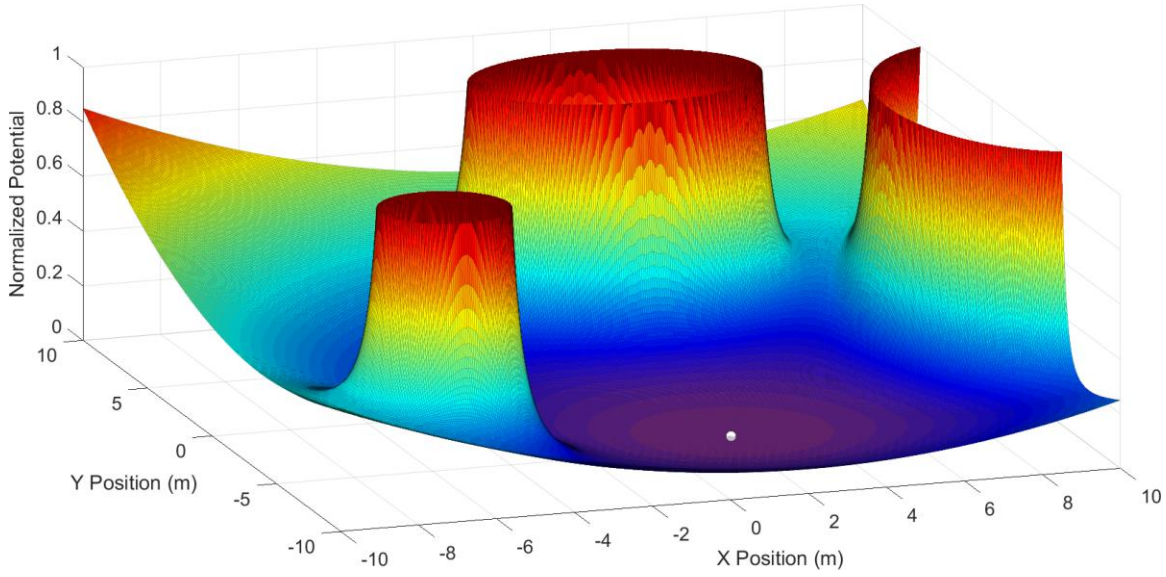

Figure 14. Example Total Potential Field with Goal at Coordinates $(2, -4)$

In general, as with the attractive force vector, the repulsive force vector $\vec{F}_{rep}$ was generated as the negative gradient of the repulsive potential field

$$\vec{F}_{rep} = -\vec{\nabla} U_{rep} \tag{6}$$

and was used to force the AGV away from obstacles it encounters [36]. As the AGV maneuvered throughout its environment, the LRF was continually scanning and publishing new laser ranges to all objects within its field of view. The repulsive force vector implemented for this thesis research was a summation of the repulsive force calculated from each of the laser returns as it scanned the environment. For this implementation, the maximum and minimum angles (with zero referenced along the forward axis of the LRF) were set to 128.75 degrees and -129 degrees, respectively, providing 257.75 degrees of laser scan coverage in front of the AGV. The LRF angular resolution of 0.25 degrees resulted in 1,032 laser ranges per scan. The angular resolution of the laser scanner and the index value *i* of each laser return were used to assign an angle to each laser range relative to the forward axis of the robot. Each laser range was used in Equation (5) to generate a repulsive potential, and the repulsive force vector at each index $\vec{F}_{repi}$ was calculated and rotated into the robot frame, based on the angle of the return, using

$$\vec{F}_{repi} = \eta \left(\frac{1}{\rho_i} - \frac{1}{\rho_0} \right) \left(\frac{1}{\rho_i^{\,2}} \right) \begin{bmatrix} -\cos(\theta_i) \\ -\sin(\theta_i) \end{bmatrix}, \tag{7}$$

where η is a positive scaling factor, ρ_i is the laser range at *i*, ρ_0 is the distance of influence, and θ_i is the angle of the laser return at *i*. When the distance to the obstacle is less than ρ_0, the repulsive force for that index is set to zero. The total repulsive force vector at point *q* is then calculated as the sum of all of the force vectors at each index

$$\vec{F}_{rep}(q) = \sum_{i=1}^{1032} \vec{F}_{repi} \tag{8}$$

and added to the attractive force vector to generate the total force vector at point q $\vec{F}(q)$ used to control the robot

$$\vec{F}(q) = \vec{F}_{att}(q) + \vec{F}_{rep}(q). \tag{9}$$

Once the magnitude of the total force vector was calculated, it was converted into a translational velocity via a scaling factor. The direction of the total force vector was converted into an angular velocity via a scaling factor and, along with the linear velocity, was published by the MATLAB Node on the /cmd_vel topic for controlling the robot's translational and rotational speed.

D. LOCAL MINIMUM

One of the major limitations of an artificial potential field algorithm is that it can generate local minima in the total potential field where the AGV can become trapped. The attractive potential field itself has only one minimum, at the goal location, but once the repulsive potential field is added, local minima may be generated. These minima can occur when an obstacle, or a collection of obstacles, exist between the AGV and the goal. These obstacles create a relatively high potential in front of the AGV, and localized areas of low potential occur. At a local minimum, the robot is unable to proceed to the goal without additional control logic for escape. The problem of local minima is two-fold—the AGV must be able to detect when it is trapped in a local minimum, and it must be able to escape to avoid being trapped again in the same minimum once it resumes its normal navigation routine.

An example total potential field in which a local minimum exists is illustrated in Figure 15. The L-shaped obstacle present in this field creates a local minimum on the opposite side of the obstacle from the goal; although, this minimum is hidden in the figure by the high potential of the object. Figure 16 is a contour plot of the same total potential field, and the local minimum is visible at a position of $(-1.3,\ 5.8)$. The minimums of this total potential are plotted as black asterisks in Figure 16, and if the AGV were to move to either of these locations, it becomes trapped. The global minimum for this example potential field is at the goal location of $(2,\ -4)$, as evidenced by the dark purple shading around the goal vice a lighter blue color around the local minimum behind the obstacle. The goal for the AGV was to reach the global minimum at the goal while being able to escape from any other local minima it encountered along the way.

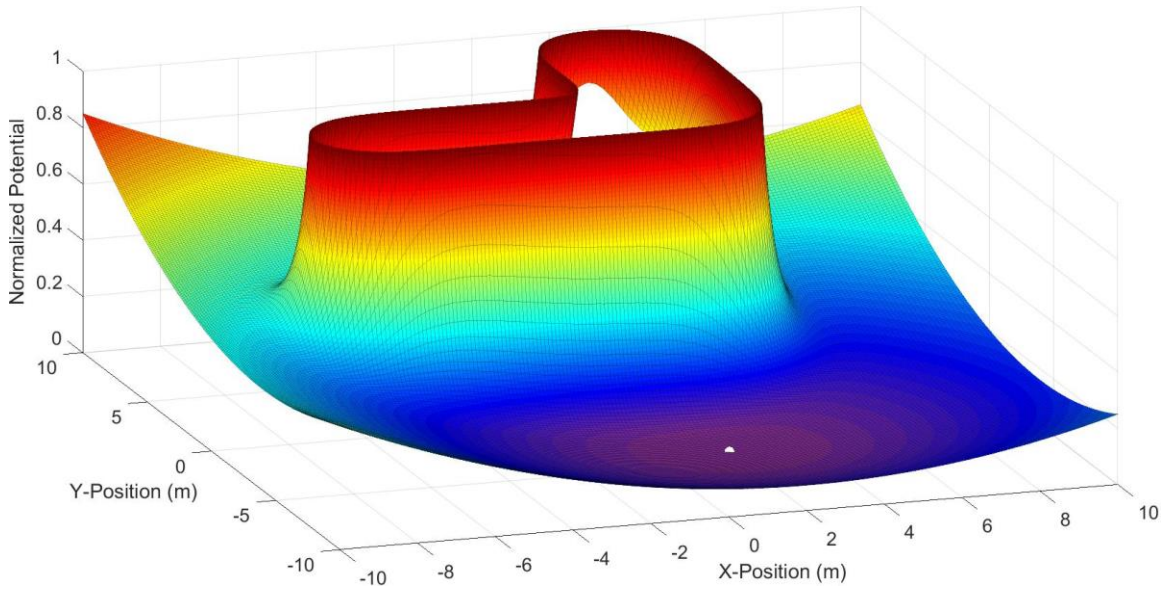

Figure 15. Example Total Potential Field with Additional Local Minimum

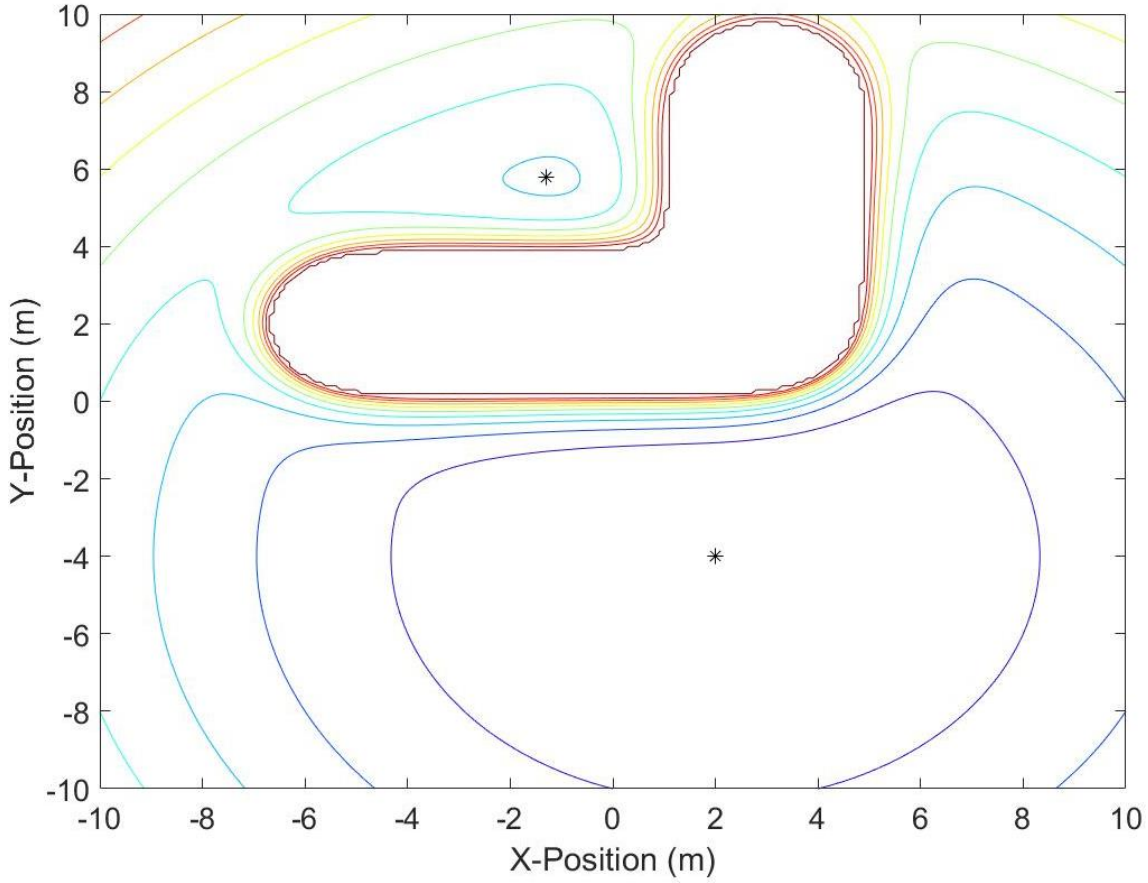

Figure 16. Contour Plot of Total Potential Field with Additional Local Minimum

Several techniques for escaping local minima have been developed, and one of the simplest methods is the wall-following method. In this method, the robot moves parallel to the obstacle's surface and continues to proceed around the obstacle until a clear path to the goal is achieved. For this thesis research, a total velocity parameter was continually evaluated to determine if a local minimum was encountered. If the total velocity of the robot—defined as the combined magnitude of the linear and angular velocities—fell below a threshold value, the robot transitioned into a wall-following mode of navigation from its normal potential field mode. The transitions between navigation modes are discussed in more detail in Chapter V.

Once in wall-following mode, the total repulsive force vector was used to determine the direction of the obstacle relative to the robot. A constant linear velocity was utilized during wall-following mode, and an angular velocity was calculated continuously to maintain the total repulsive force vector between 80 and 100 degrees from the robot's forward axis. The direction that the robot turned initially was based on the direction of the repulsive force vector prior to transitioning into wall-following mode. If the repulsive force vector was anywhere to the left of the robot's forward axis prior to the transition, the robot turned right, and vice versa. This allowed the robot to maneuver parallel to the obstacle's surface, regardless of its shape, and continue around the obstacle to escape the local minimum. Once the distance to the goal position began to decrease, the robot transitioned back to its potential field mode of navigation to continue towards the goal. This is discussed in detail in Chapter V.

The algorithms discussed in this chapter allowed the AGV to detect and avoid obstacles in unknown configurations and proceed toward a goal position. A wall-following algorithm was developed to overcome the problem of local minima and was used effectively to escape and continue toward the goal. The localization methods used to determine the positions of both the AGV and the goal within the environment are discussed in Chapter V. The overall algorithm used to transition between the different modes of navigation used by the AGV is also presented as well as the actions of the robot when it detected obstacles too close for avoidance.

V. LOCALIZATION AND CONTROL STRUCTURE

The algorithms discussed in Chapter IV provided the AGV with a robust method for detecting and avoiding obstacles, but its utility was significantly limited without accurate localization information. Without a reliable source of localization data, the AGV was not able to reach its desired goals effectively in dynamic outdoor environments. Two methods of localization were employed during the experimentation phase of this thesis research, which were based on the availability of a GNSS signal. They are the focus of the first part of this chapter.

With accurate localization of both the AGV and the goal, the algorithms presented in Chapter IV were used to affect the attainment of the goal by the AGV. The construct utilized to organize and transition between the various algorithms developed for this thesis research was the switch-case structure in MATLAB. This structure allowed the AGV to operate in one of many modes—or cases—depending on what it sensed in its environment. In the second part of this chapter, we discuss the main MATLAB script, provided in Appendix A, which implements the switch-case structure. This script facilitated the control logic for entering into and transitioning between the various modes of operation of the AGV as it progressed toward its goal.

A. LOCALIZATION

Localization of the robot's position and goal was accomplished using two different techniques during development and testing of the system. Throughout the initial phases of experimentation, the P3-AT's onboard encoders were used to localize the robot's position via dead reckoning. The robot's position, orientation, and velocity were calculated and published to the ROS network by the RosAria Node based on the number of counts registered from each motor's encoder. The RosAria Node published this information on a /RosAria_Node/pose topic, which included the robot's X and Y position (the Z position was forced to zero, as the robot only moved on flat surfaces), its orientation as a quaternion, and its linear and angular velocities along the X, Y, and Z axes. The origin of the local X-Y world reference frame and its orientation were

established using the robot's initial position and orientation when the robot was powered on. The positive X-axis was initialized along the robot's forward axis, and the positive Y-axis was oriented out the left side of the robot. This resulted in a positive Z-axis up through the top of the robot.

As the robot maneuvered throughout its environment, the RosAria Node published an updated message at a rate of 10 Hz to the ROS network. The RosAria Node also published a coordinate transform on the /tf topic to provide information about the coordinate transformation between the robot frame and the local world frame for use by other nodes in the ROS network. Goal localization during this phase of testing was accomplished by establishing a relative goal based on the initial robot position as it was powered on. For example, a goal location of (10, 0) was located 10 meters directly in front of the robot when it was first powered on. A goal of $(5, -6)$ was located five meters in front of the robot and six meters to the right of the robot. This method of localization—using dead reckoning and relative goal positions—was simple to implement and facilitated rapid development of navigation algorithms. The major disadvantage of using this method was that wheel slippage over the ground resulted in significant cumulative errors in position. As the wheel slipped, the encoders continued to count ticks, but the robot was not actually moving at the same rate over ground. This effect resulted in undesirable position error; however, the effect was minimized in the indoor environment where the ground surface was smooth and flat. Over loose terrain encountered outdoors, position errors built up rapidly. The errors present in localization were not significant enough to effect the development of the navigation algorithms, however, and this method was sufficient during the initial testing phase, as discussed in Chapter VI.

Dead reckoning via motor encoders, while accurate over short distances, was not robust enough to implement in the final system. During the outdoor phase of testing, we accomplished localization using data from the GNSS/INS. This sensor fused data from its internal IMU with fix information obtained from its integrated GNSS antenna via an EKF. Once initialized, the EKF provided an estimate of the robot's position and orientation through the INS Node, which published this information on the /nav/odom topic to the ROS network. After the system was powered on, the EKF started providing

an estimated robot position once a valid fix was obtained from the GNSS and the filter converged to a solution. The control algorithm, discussed in detail in the next section, monitored the /nav/status topic to ensure the EKF was not operating in a degraded state. If the EKF estimate was degraded due to either a bad GNSS fix or high covariance in the estimation, the control algorithm reset the filter once a good fix was obtained. Goal localization was accomplished using a table of waypoints read by the control algorithm, discussed in the next section, and published to the ROS network on the /nav/goal_odom topic.

The robot and goal localization data published on the /nav/odom and /nav/goal_odom topics were represented in latitude and longitude as defined by the WGS84 standard. In order to use this data for navigation in a common local reference frame, each was transformed into UTM coordinates referenced to a local datum parameter (defined as 36.595 degrees latitude and −121.875 degrees longitude, on the Naval Postgraduate School campus). The GeoNav Transform Node performed the transformation of the robot's position, while the Goal GeoNav Transform Node transformed the goal position, resulting in local X and Y coordinates for both the robot and the goal. These coordinates were then used for navigation, and this technique was successful in outdoor testing, as described in Chapter VI.

B. IMPLEMENTATION OF THE CONTROL ALGORITHM

A MATLAB script, provided in Appendix A, was used to initialize parameters and control the AGV through specific operating modes. Once the MATLAB Node was generated and connected to the ROS network, by using the rosinit command in MATLAB, this script was run to control the AGV as it maneuvered toward its goal. A switch-case structure was used to transition between the operating modes, which allowed for flexibility and modularity of the script during the development process.

The control script first established the ROS subscribers and publishers used by the MATLAB Node and then imported algorithm parameters and goal coordinates from a text file. The use of an external text file allowed the goal and specific parameters to be adjusted independently of the control script. Ten goal waypoints and other important

operating parameters were included in the text file for adjustment during the testing phase of this thesis research. Once these operating parameters were imported and initialized, the control script prompted the user for which waypoint to assign as the goal. The goal coordinates were then published by the MATLAB Node for use by the Goal GeoNav Transform Node, and the control script checked the status of the GNSS/INS EKF estimation. If the estimated position was valid, the script continued to define other variables and parameters; if it was not valid, the script reset the EKF.

The main portion of the control script consisted of four distinct cases, or modes, of operation for the AGV nested inside a while loop to continue the execution of the script until the goal was reached. A state diagram illustrating the different cases and the conditions required to transition between cases are shown in Figure 17. A case variable *c* was used to differentiate between the modes—the potential field mode, the wall-following mode, and two emergency escape modes—and each mode was numbered, one to four. At the beginning of each iteration of the while loop, updated values for laser ranges, laser angles, goal position, GNSS fix, and EKF position estimation status were obtained. Prior to determining which mode of operation the AGV would enter, the position estimation status was checked, and the EKF was reset, if necessary. The estimation filter was only reset if there was already a valid estimated position in the past (the EKF had completed the first initialization phase), the estimation was now degraded, and a valid GNSS fix was currently available. This part of the control script allowed the AGV to recover from a loss of GNSS signal or a disruption in the estimation filter.

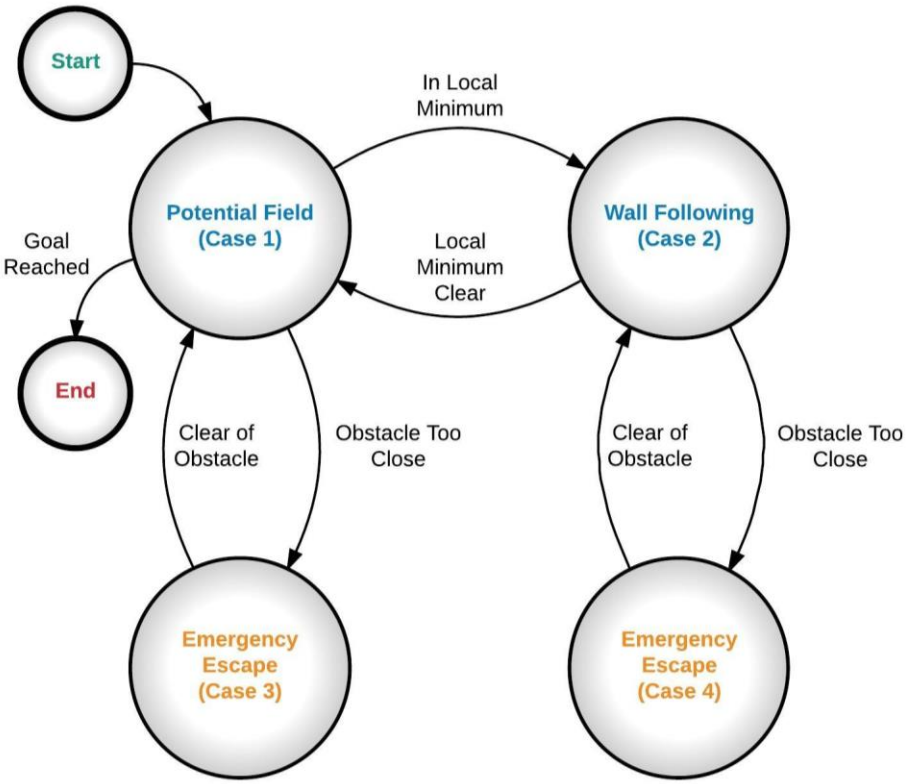

Figure 17. State Diagram of Control Logic

The case variable *c* was initialized to one, so the first time through the control script, the AGV entered the potential field mode of operation. This was the main mode of operation for the AGV, and it provided both obstacle avoidance and navigation to the goal position under normal operation. In each iteration of the potential filed mode, the distance to the goal position was calculated. If the goal had not been reached, the potential field algorithm developed a total force vector that was used to calculate desired linear and angular velocities for the AGV. If the goal had been reached (defined as within 0.5 m of the goal position), the AGV stopped, a message was printed to the MATLAB command line, and the script broke out of the while loop and terminated.

Transitions between modes of operation were accomplished by checking various parameters and setting a value for *c* at the end of each case. This control scheme allowed for the evaluation of the AGV's current operating conditions during each iteration of the while loop and the execution of a different mode of operation if specific conditions were

met. At the end of the potential field case, conditions for entering the wall-following mode (case two) and the first emergency escape mode (case three) were evaluated. If the magnitude of the total velocity of the robot was below a threshold, indicating that the robot was at a local minimum, and the robot had not yet reached the goal, *c* was set to two. If an obstacle was detected within a minimum range required for effective avoidance, set to 0.5 m, *c* was set to three. If neither of these conditions existed, *c* was set to one, and the robot continued in potential field mode.

The wall-following mode of operation was used to escape from a local minimum, and the variable *Dcount* was used as a counter to help determine when to exit this mode and return to the potential field mode. In order to escape from a local minimum, the robot maneuvered around the obstacle for a sufficient distance such that it would not become trapped again in the same minimum once it returned to potential field mode. The distance to the goal position was calculated during each iteration of wall following, and each time that the distance to the goal was less than the previous iteration, *Dcount* was incremented by one. At the end of each iteration of wall-following mode, conditions for returning to potential field mode (case one) or the second emergency escape mode (case four) were evaluated. When *Dcount* reached a threshold value, indicating that the robot had been moving toward the goal for a sufficient number of iterations and had escaped the local minimum, the case variable *c* and the counting variable *Dcount* were set to one. If an object within 0.5 m had been detected, *c* was set to four. If neither of these conditions were met, *c* was set to two, and the robot continued in wall-following mode.

The emergency escape mode of operation was a safety algorithm designed to prevent the AGV from impacting obstacles that were too close for avoidance. Two separate cases, case three and case four, were used in the control script to implement the emergency avoidance algorithm depending on which mode of operation the AGV was operating in just prior to entering emergency escape mode. Both of these emergency modes executed the same algorithm, and the only difference between them was to which mode of operation they returned when the algorithm was completed. The emergency escape algorithm consisted of the AGV immediately stopping for five seconds and then backing up slowly for four seconds. This algorithm, although simple, allowed time for

dynamic obstacles to move out of the way of the AGV and allowed enough room for the AGV to maneuver around the obstacle if the object was static. After each execution cycle of the emergency avoidance mode, laser ranges were updated, and the distance to the nearest obstacle was calculated. If an object was still detected within 0.5 m, *c* was set to either three or four (corresponding to the current instance of the emergency escape mode), and the algorithm was repeated. If all objects were outside of 0.5 m, *c* was set to one if the mode prior to the emergency was the potential field mode, or *c* was set to two if the prior mode was wall following.

These modes of operation allowed the AGV to change its behavior based on what it encountered in its environment and facilitated the attainment of the goal position. In each case, the required linear and angular velocities of the AGV were calculated. To prevent large, abrupt changes in commanded velocities (in all cases except for the emergency escape cases), the new velocities were filtered with the current velocities via scaling factors in order to smooth out the accelerations on the vehicle. After the execution of each case, and prior to obtaining new information from the environment, these velocities were published by the MATLAB Node on the /cmd_vel topic to control the AGV chassis.

The system described in Chapters II through V was developed utilizing many different techniques including simulation, experimentation, and testing. The methodology used to design and test this system is outlined in Chapter VI, along with specific results during each phase of testing. Each stage of development is presented, ranging from individual sensor characterization and experimentation to integrated system testing in varying outdoor environments.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. RESULTS

After the initial system integration was completed and the underlying ROS software framework was implemented, the final three phases of the system development were accomplished. These were to 1) carry out benchtop experimentation and testing of the individual system components, 2) perform simulations of the potential field and control algorithms in Gazebo, and 3) conduct real-world indoor/outdoor testing with the AGV. The results obtained from these final three phases are presented in this chapter.

A. LABORATORY BENCHTOP EXPERIMENTATION RESULTS

The purpose of the initial laboratory experimentation phase was to characterize the available sensors, specifically the LRF and the sonar, to determine if the LRF alone was sufficient for obstacle detection and avoidance. This phase also evaluated the effectiveness of the GNSS/INS and determined if its output, specifically the magnetometer heading, was affected by the operation of the chassis motors and the SlimPRO PC. The last part of this phase of testing involved performing a stress test on the SlimPRO's processor and the onboard battery bank of the P3-AT to ensure these components were able to handle the demands of follow-on system testing.

1. Individual Sensor Performance

The integration of the LRF into the system was tested during this phase by verifying the effectiveness of the heat sink plate over extended laser operation and by confirming that the laser was providing accurate and timely scan data to the ROS network. Once system integration was tested, the characteristics of the LRF, to include maximum range, angular resolution, and field of view, were determined with the sensor mounted in place on the robot chassis. The laser scan data was visualized using rviz, and it was used to determine the specific operating characteristics of the LRF. The typical laboratory environment used during this phase of testing is pictured in Figure 18.

Figure 18. Typical Laboratory Testing Environment

The objects in the laboratory environment that were detected by the LRF are represented in Figure 19. The returns in rviz are shown from a similar perspective as Figure 18, and they demonstrate the high angular resolution of the LRF. Large objects, such as the storage cabinets and walls, are immediately apparent from an examination of the laser returns, and even thin objects, such as table legs, chair legs, and the broom stick, were all detected in this case. The maximum range of the laser was verified to be greater than the manufacturer's stated maximum range of 30 m, as objects at ranges up to 35 m were reliably detected in the laboratory setting. While the laser was limited to a 2D plane of detection and was not able to scan behind the AGV, its high resolution and fast refresh rate resulted in the accurate detection of all objects within its plane of view.

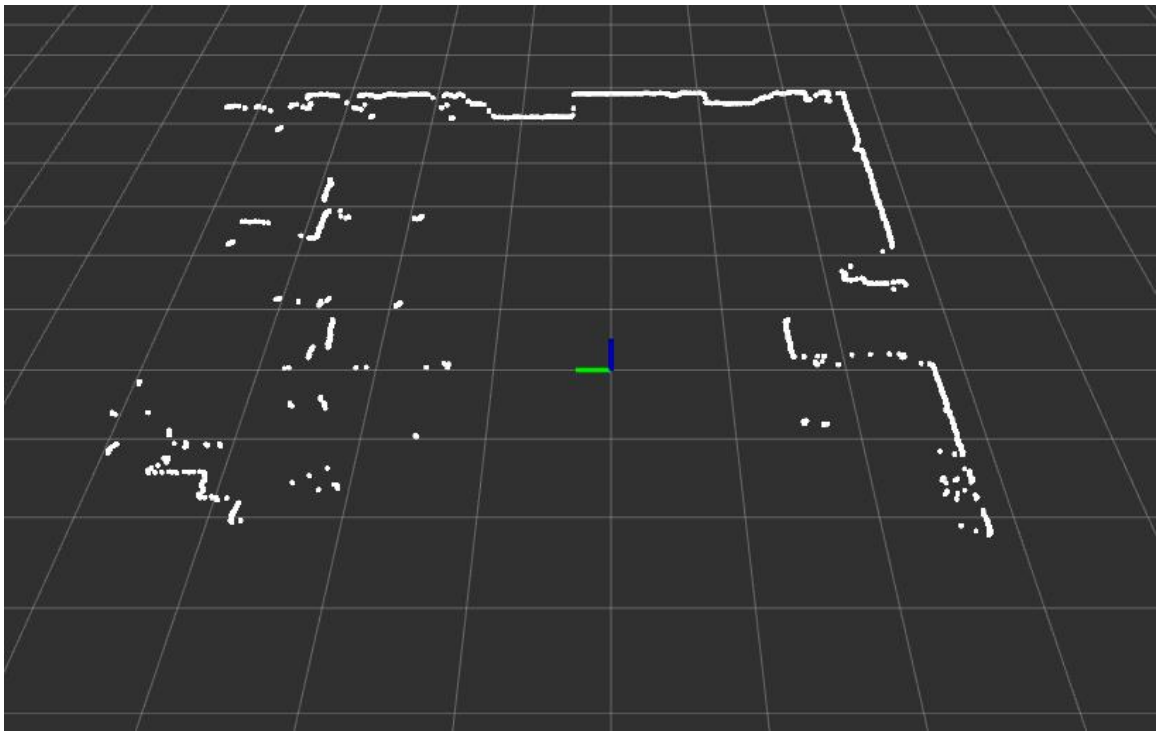

Figure 19. rviz LRF Data from Laboratory Environment

The onboard sonar sensors were tested and characterized by utilizing the RosAria Node, which published sonar range data to the ROS network. This range data was visualized in rviz, and the characteristics of the sonar transducers, to include maximum range, angular resolution, and responsiveness, were determined. The sonar returns from the environment illustrated in Figure 18 are shown in Figure 20. The differences between the sonar data and the LRF data are immediately apparent when comparing Figures 19 and 20. The most striking difference between the two is the angular resolution. The drastically lower angular resolution of the sonar transducers resulted in a significant decrease in object detection from the environment, with objects only being detected when directly in front of one of the 16 transducers mounted around the chassis. When encountering large objects, such as those large enough to be detected by multiple sonar sensors at the same time, the ranging information from the sonar was relatively consistent. When faced with smaller objects, however, the sonar ranges to the object were less reliable, and the sensor would often not detect the object for multiple scans before again getting a return. This unreliable behavior, coupled with a relatively slower response

time, resulted in many objects going undetected by the sonar sensor in the laboratory environment that were detected by the LRF.

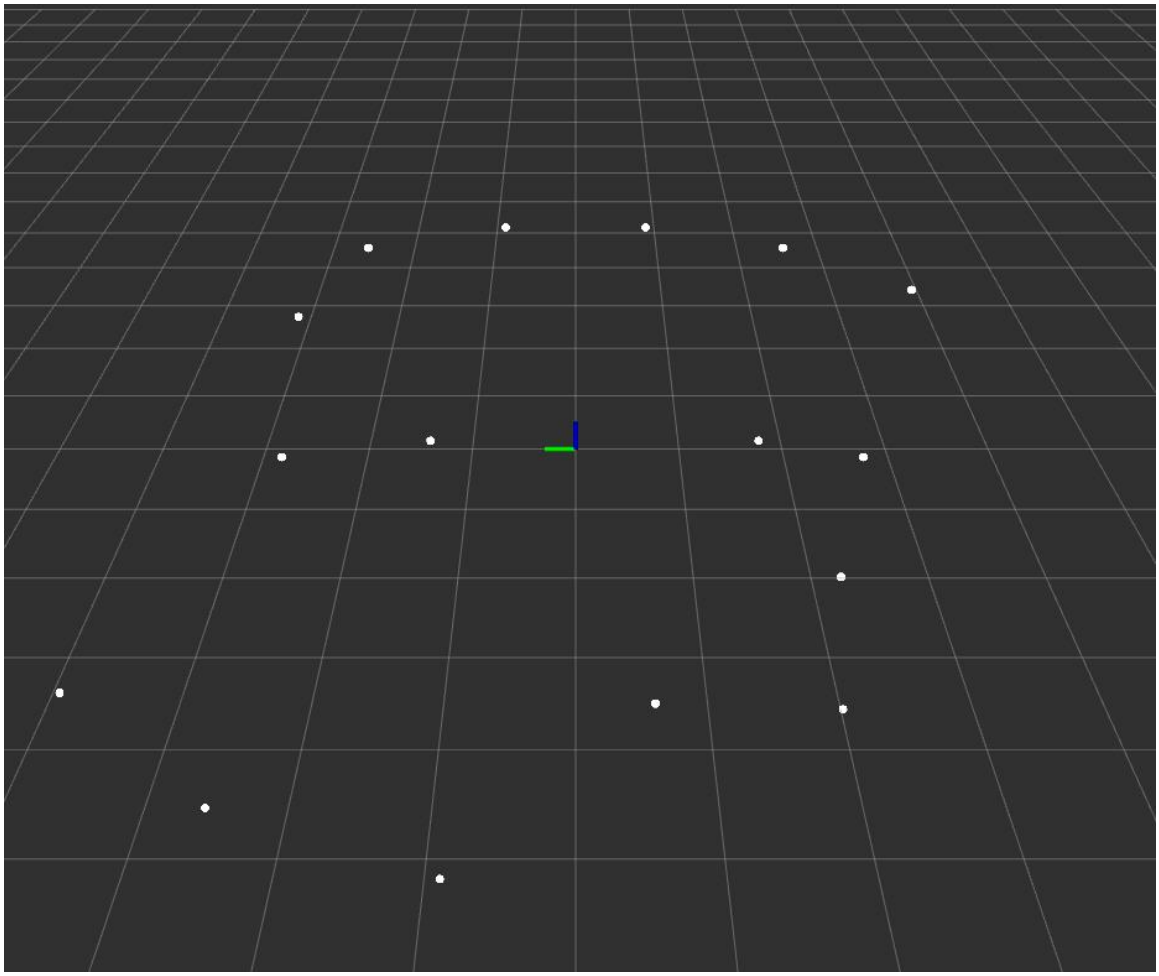

Figure 20. rviz Sonar Data from Laboratory Environment

The maximum range of the sonar sensors was five meters, and if any sensor did not receive a return during a scan, it reported a value of five meters back to the ROS network. This is why each sensor has a return in Figure 20, even though there were no actual objects within five meters for many of the sensors. The sonar was able to detect some obstacles slightly above and below the plane of transducers due to the spreading of the sound wave as it propagated, but the ranging information for these types of obstacles was inconsistent and unreliable. The sonar sensors were able to detect major obstacles

within the laboratory environment and were able to detect objects behind the AGV, but the low resolution and slow response time severely affected the fidelity of the environmental data provided to the ROS network.

The initial testing and configuration of the GNSS/INS was accomplished using LORD MicroStrain's MIP Monitor software. This software was used to connect to the GNSS/INS and verify that all of the individual sensors were operating correctly. The orientation of the output data was also verified using MIP Monitor. The sensor outputs its data with the positive X-axis in the North direction, the positive Y-axis in the East direction, and the positive Z-axis down toward the Earth, or North-East-Down (NED). This orientation differs from the ROS standard reference system of East-North-Up (ENU). In the ENU coordinate system, the positive X-axis is in the East direction, the positive Y-axis is in the North direction, and the positive Z-axis is up, away from the Earth. The INS Node performed the conversion of the GNSS/INS data from the NED to the ENU coordinate frame by swapping the X and Y data and inverting Z. The sensor was mounted on the robot chassis with its positive Y-axis toward the front of the robot and its positive X-axis oriented to the robot's left in order to provide the same local X-Y world frame, after the INS Node conversion, as the one used during encoder-only localization.

The MIP Monitor software was also used to determine if the operation of the P3-AT chassis motors and the SlimPRO PC affected the magnetometer heading outputted by the GNSS/INS. The preliminary assumption prior to testing was that these effects would have a minimal impact on the sensor's performance based on the findings of Bachmann et al. in [38]. The experiments performed in [38] indicated that while magnetic distortion occurred when the sensor was moved close to metal objects, this distortion was not exacerbated by the operation of the robot's motors. The assumption in this thesis research, which was that the chassis motors did not affect the operation of the GNSS/INS, was tested by recording and analyzing sensor output data during different modes of operation of the AGV. Data sets were recorded with both the chassis motors and SlimPRO PC running, as well as with both components powered off. For each of these operating modes, data was recorded with the sensor mounted at two positions above the

chassis—in the position illustrated in Figure 6, and at a position 32 cm higher, mounted on a plastic arm—resulting in four sets of data for analysis. Histograms of the results are shown in Figures 21 and 22. The mean values from each data set are plotted as a red line in each subplot of Figure 21, and the blue dashed lines represent one standard deviation from the mean on either side. The histograms from each mounting position of the GNSS/INS are shown plotted on the same axes in Figure 22 for comparison.

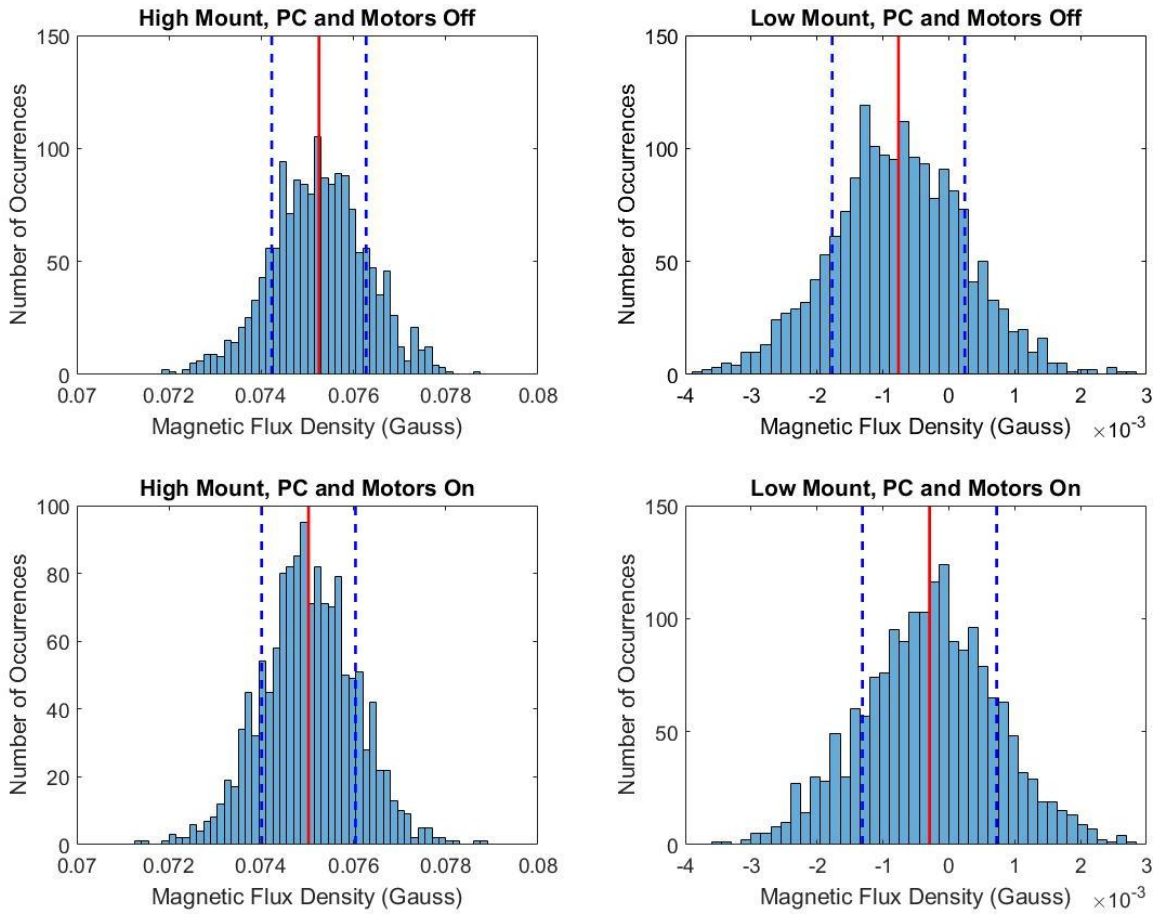

Figure 21. Histograms from Magnetometer Testing under Various System Conditions

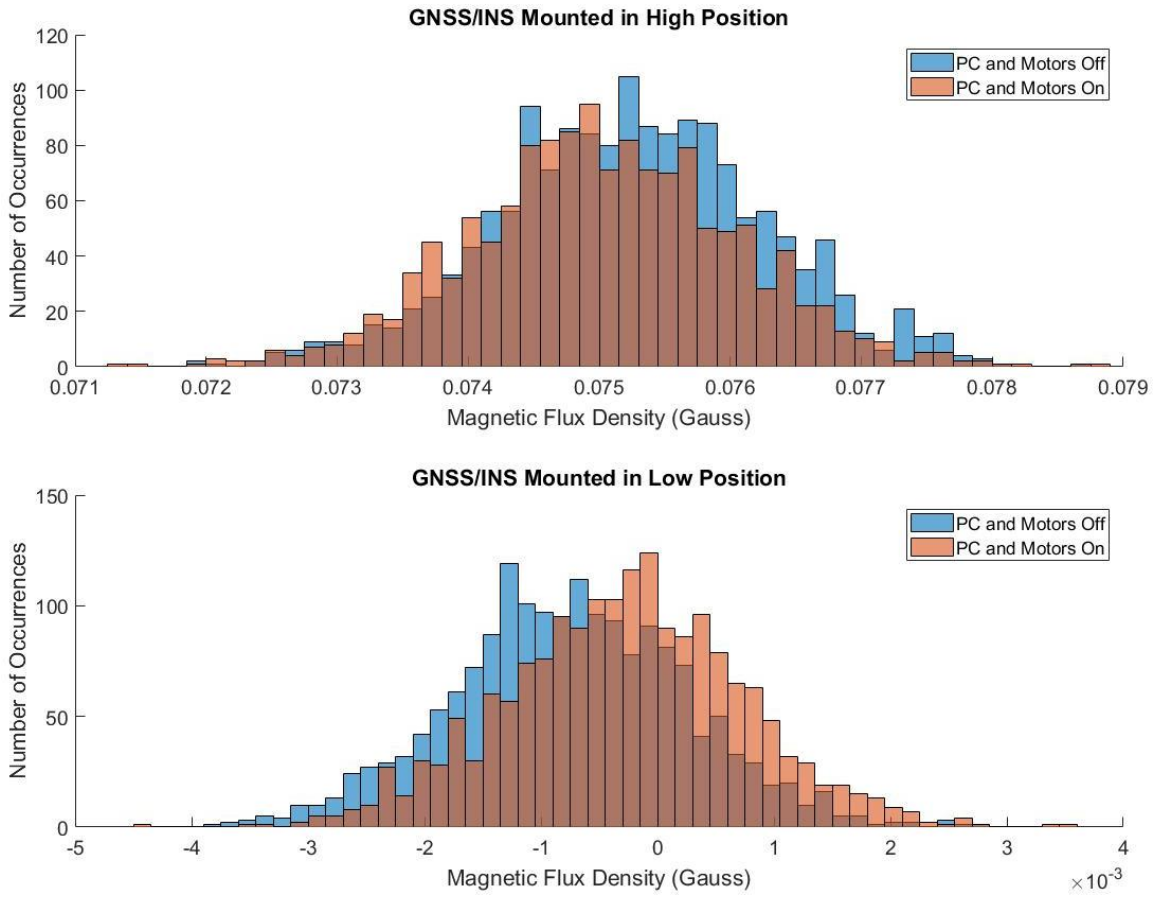

Figure 22. Histogram Comparison of Magnetometer Data

The mean and standard deviation for each case was calculated for comparison, as well as the difference between the data during the on and off states of the AGV. The calculated data is provided in Table 1. An analysis of the test data indicated that operation of the motors and SlimPRO PC had a negligible effect on GNSS/INS operation, and the preliminary assumption was confirmed.

Table 1. Magnetometer Testing Data

High Position (All Units in Gauss)			**Low Position (All Units in Gauss)**		
State	**Mean**	**Standard Deviation**	**State**	**Mean**	**Standard Deviation**
On	0.07503	1.018×10^{-3}	On	-0.00029	1.019×10^{-3}
Off	0.07525	1.023×10^{-3}	Off	-0.00076	1.007×10^{-3}
Difference	2.248×10^{-4}	5.66×10^{-6}	Difference	4.718×10^{-4}	1.21×10^{-5}

The integration of the GNSS/INS into the system was tested by verifying that the sensor provided continuous raw IMU data, GNSS fix data, EKF status, and an estimated AGV position to the ROS network, and the service required to reset the EKF was tested. The GNSS antenna was placed on a windowsill in the laboratory in order to obtain a GNSS fix for testing purposes, and the GNSS/INS was able to provide continuous position estimates for the AGV once a valid fix was obtained.

The sensor characterization and testing conducted during this phase of the thesis resulted in the determination that the LRF was sufficient as an environmental sensor for object detection without the addition of sonar sensor data into the potential field algorithm. The unique information provided by the sonar sensors, which was not already provided by the LRF, was limited and inconsistent and would unnecessarily complicate the algorithm. This testing also verified that the GNSS/INS provided continuous position estimates for the AGV to allow for effective localization in outdoor environments, and the sensor suite was ready for further testing in the final phase of development.

2. Component Stress Testing

Once the sensor suite was configured and tested individually, stress testing was conducted on the SlimPRO's processor and the P3-AT's battery bank. The processing power of the SlimPRO PC was tested to ensure that the computer would not become overloaded during AGV operation. The AGV was placed on a stand so that the drive wheels were off the ground, and the system was started. With the complete ROS network running, including all sensors, and MATLAB executing a control script to drive the wheels, the loading on the PC's processor was monitored via the built-in system monitoring software in Ubuntu. On average, the dual-core processor only utilized approximately 66 percent of its capacity during this test period, representing the anticipated load condition during normal operation of the system. Processor testing continued with additional loads placed on the PC to determine the maximum load allowed by the SlimPRO. In addition to the loads running during the previous test, an instance of rviz was added. With rviz displaying the real-time LRF data, the processor operated at approximately 74 percent capacity. The processing load was increased to an

average of 94 percent capacity by starting a Gazebo simulation involving three dynamic structures. To push the processor to its maximum capacity (100 percent), a ROS topic visualizer was started to display the current information published on every topic in the ROS network. Even with the addition of significant loading on the system beyond the normal anticipated load, the SlimPRO was able to process all of the sensor data and execute the MATLAB script. A slight delay in AGV responsiveness was noted during the last portion of this test (when the processor was operating at maximum capacity), indicating that the limit of the SlimPRO's processing power had been reached. Any additional load on the system beyond this point would start to affect AGV operation significantly. The processor stress testing provided confidence in the ability of the SlimPRO to execute the control script and process all of the sensors with enough reserve capacity for expansion in the future, if required.

The next stress test conducted was an endurance test of the AGV's battery bank. After a full charge and verification of each battery's voltage independently, the combined battery bank was tested under a continuous load until the bank voltage dropped to 11.0 VDC—the minimum voltage allowed by the P3-AT for motor and microcontroller operation. With the AGV on a stand and its drive wheels off the ground, electrical load was created by running the entire sensor suite, conducting wireless communications to an external laptop via SSH connection, and all drive motors running at the equivalent translational speed of 0.5 m/s continuously. The RosAria Node was used to monitor the battery bank voltage during the test, and the test was stopped once the node indicated 11.0 VDC. The AGV's batteries were able to power the system under this large load for over three hours before reaching the minimum voltage. The battery endurance test indicated that the system was able to operate for a significant period after a full charge, especially when the AGV was not running at full load continuously.

The component stress testing verified that the system was able to function as desired without being limited by batteries or the computer's processing power under normal operation. The initial phase of laboratory experimentation and testing verified the operation of each sensor individually and the system as a whole. The hardware for the

AGV was ready to move on to the final phase of testing after the control algorithms were tested during the simulation phase.

B. SIMULATION RESULTS

The potential field algorithm and the overall control algorithm, described in Chapters IV and V, were developed and tested in a simulated environment using Gazebo. To simplify the development process, the localization method used for simulation was dead reckoning via the RosAria Node as described in the first part of Chapter V. This method of localization was adequate in simulation, as the phenomenon of wheel slippage was not simulated in Gazebo. The basic environment depicted in Figure 10 was created for initial verification of the control script, and the four spherical obstacles provided a simple test for the initial potential field algorithm. An early simulation with the robot maneuvering around one object is shown in Figure 23. In this simulation, shown from a top-down orthographic view, the robot started from the origin in the green box and navigated towards the goal position, which was at the yellow dot shown at coordinates (10, 10). The trail of the robot as it maneuvered toward its goal is shown in red, and the blue rays indicate the laser returns of the LRF.

The robot was able to effectively detect and avoid the spherical obstacle in its path, and the potential field algorithm was effective in navigating the robot to the goal location. This simulation environment was used to fine-tune various parameters and constants of the control algorithm, and once the robot was able to efficiently reach the goal with these simple obstacles, more complex environments were created to test other aspects of the control script. Gazebo allows for objects to be placed into the simulation environment as the simulation is running, so dynamic obstacles were simulated by placing objects in the path of the robot as it was moving. Placing obstacles dynamically allowed for the testing of the emergency avoidance mode of operation by placing obstacles within 0.5 m of the robot.

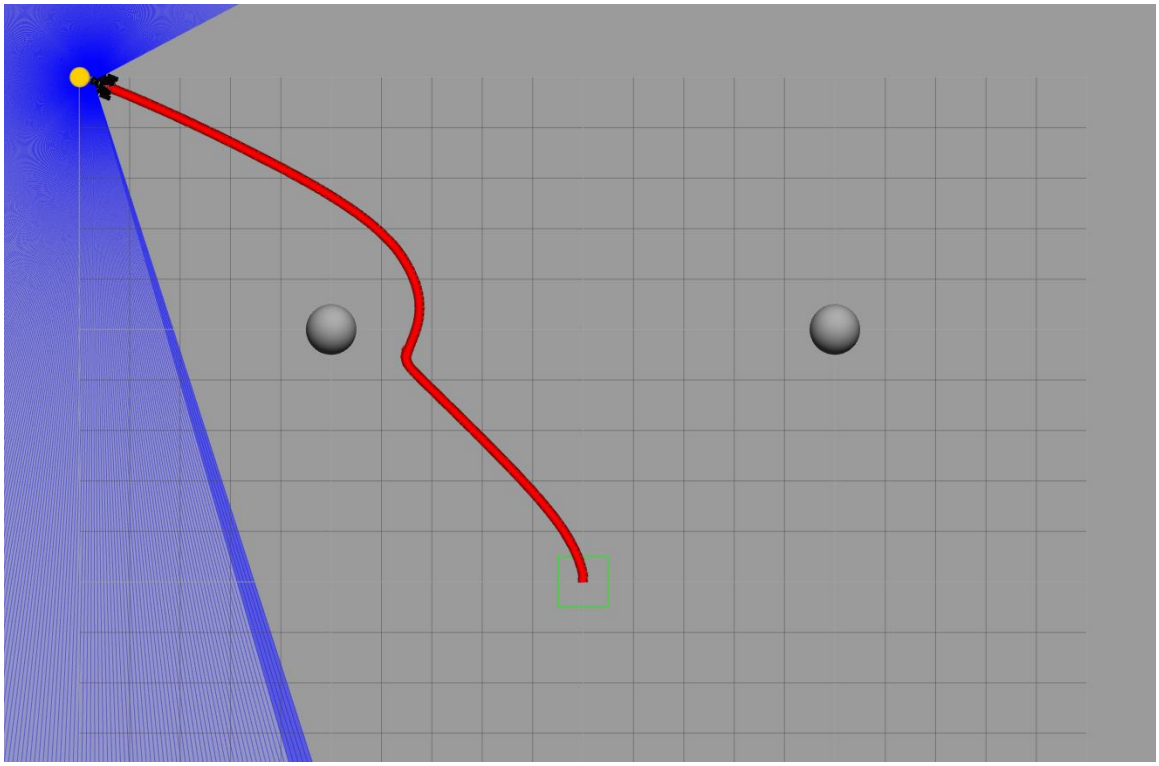

Figure 23. Simulation with One Obstacle and Goal at (10, 10)

More obstacles were added to the simulated world to test the potential field algorithm in different environments, and once the emergency avoidance algorithm was verified, the environment shown in Figures 24 and 25 was created to test the robot's ability to escape from a local minimum using wall following. In Figure 24, the robot started and ended at the same locations as in Figure 23, except that now the corner of the wall obstacle created a local minimum along the robot's path. In this case, the simulation showed that the robot was able to identify that it was in a local minimum and transitioned to wall-following mode, shown as the green portion of its path.

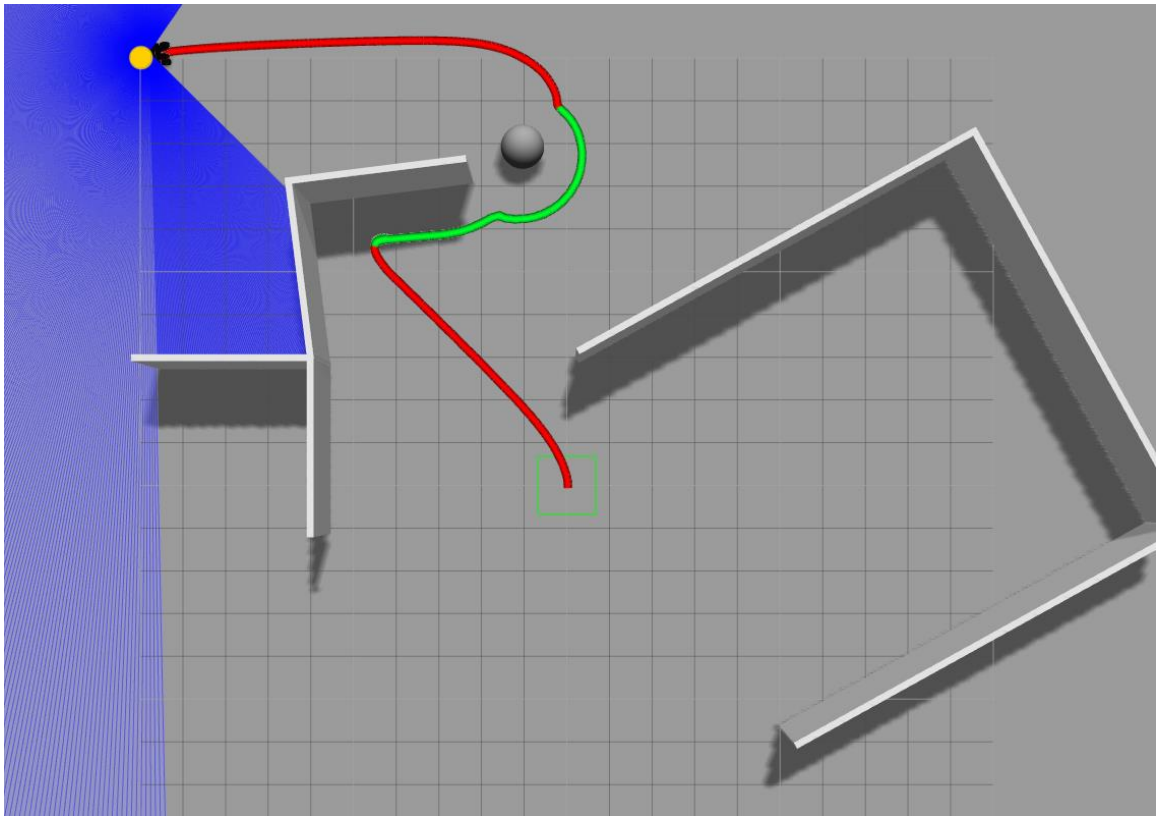

Figure 24. Simulation with Multiple Obstacles and Goal at (10, 10)

The robot continued in wall-following mode, moving along the rest of the wall and then around the spherical obstacle, until the distance to the goal started to decrease. Once the robot had moved toward the goal again, indicating that it had escaped from the local minimum, it transitioned back to potential field mode of operation—the red path—and was able to reach the goal. The turning logic, which determined the direction the robot turned when it transitioned into wall-following mode, was verified by moving the goal position within the same environment.

The path of the robot with the goal position at $(10, -10)$ is shown in Figure 25. In this case, the robot entered wall-following mode after becoming trapped in a local minimum and followed the wall until it was able to head back toward the goal position. Once it was clear of the local minimum, it transitioned back into potential field mode and proceeded to the goal.

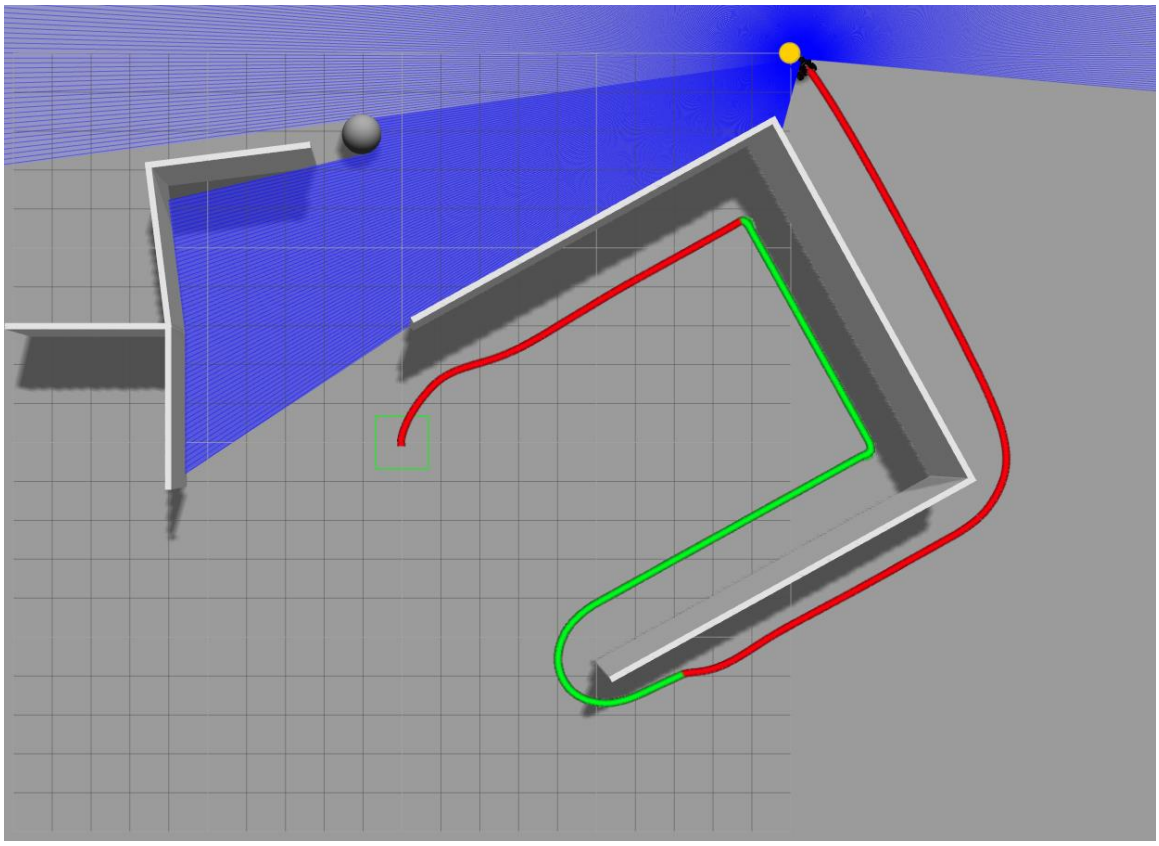

Figure 25. Simulation with Multiple Obstacles and Goal at (10, −10)

The simulation environment provided a rapid development platform for the potential field and control algorithms; it was utilized to configure the algorithm gains and constants to provide smooth movement of the robot. The ability for the robot to detect and avoid obstacles using only the LRF was verified, and the parameters developed in simulation served as a starting point for further refinement during the final phase of system development. At the conclusion of the simulation phase, the potential field, emergency avoidance, and general control algorithms were verified, and the AGV was ready to move on to real world testing in both the laboratory and in various outdoor environments.

C. REAL-WORLD EXPERIMENTAL RESULTS

With the system hardware configured and the controlling algorithms verified in simulation, the AGV was tested in both the laboratory and outdoors. Testing in the

laboratory was performed first to refine the algorithm gains and to ensure that the robot traversed at a safe speed. The controlling algorithms were tested again using the actual system hardware and were verified against both static and dynamic obstacles. Once the robot was able to navigate effectively in the laboratory environment, testing commenced in various outdoor environments utilizing the GNSS/INS for localization.

1. Laboratory Robot Testing

The localization methods and algorithm constants developed in simulation were used as the starting point for testing the AGV in the laboratory. An external laptop was used to remotely start up the system and monitor parameters during testing to determine if the AGV was operating as expected. The external laptop communicated with the SlimPRO via SSH connection over WiFi. Initial testing involved verification of the behavior observed during simulation, and navigation around simple, static objects. Dead reckoning localization through the RosAria Node was employed, as a GNSS signal was not available, and relative goal positions were used. This type of localization was effective for the relatively short distances encountered in the laboratory, and the accumulated position error from the motor encoders was minimized on the hard, smooth floor. This position error became evident after the robot had reached multiple goals without resetting, especially when the robot had to turn multiple times along its path. In these cases, the accumulated error grew to a meter or more and began to affect the attainment of subsequent goal positions. As such, the robot was reset periodically during testing to bring the position error back to zero.

Parameter gains for each of the control algorithms and the commanded linear and angular velocities were adjusted during this phase of testing to ensure that the robot operated safely with smooth movements. The dynamics of the AGV chassis and the latency of the system required slight changes in these parameters from the ones developed in simulation. Once these parameters were adjusted, more complex laboratory environments were used to test the wall-following and emergency avoidance algorithms. The AGV demonstrated the ability to transition between the potential field and the wall-following modes of operation in the laboratory, and the output filter used to prevent

abrupt changes in the commanded velocity was adjusted to provide smooth transitions between the various operating modes.

Dynamic obstacles were introduced in the laboratory to include both moving objects and people walking along side and in front of the AGV. In these cases, the system was able to detect the obstacle and move away from it to prevent a collision. If the object moved too quickly for effective avoidance, the AGV transitioned into emergency escape mode. Following the execution of the emergency escape algorithm, the system was able to transition back into either potential field or wall-following mode once the obstacle was clear and continue toward the goal.

The laboratory phase of testing and development verified the performance of the obstacle avoidance and control algorithms developed in simulation and confirmed that the AGV could operate at a safe speed prior to the outdoor testing phase. The density and variety of obstacles present in the laboratory environment provided significant challenges for the system as it maneuvered toward its goal. The AGV was not limited by terrain due to the floor's consistent, hard surface, but the density of obstacles was much higher in the laboratory than what was expected for most outdoor locations. The ability of the system to reach its goal position effectively in this type of environment demonstrated that the AGV was ready to transition to outdoor testing.

2. Outdoor Robot Testing

The final phase of development for the AGV was outdoor testing in various unstructured environments. For this phase, the localization scheme shifted from dead reckoning via motor encoders to position estimation provided by the GNSS/INS. The first part of testing performed was a verification of the localization and navigation algorithm required for accurate position estimation. Ten waypoint coordinates were used for outdoor testing, and the terrain between these waypoints included concrete, dirt, grass, and mulch. Initial outdoor testing involved commanding the AGV to travel from one waypoint to another over concrete to test the localization and navigation algorithms without the presence of obstacles. During this test, the AGV had a clear view of the sky to ensure that it was not limited by a degraded GNSS signal. The system was able to

reach the goal waypoint effectively during this test, and the AGV was able to stop within one meter of the goal reliably.

With the localization algorithm verified, waypoints were chosen to force the AGV around obstacles scattered throughout the environment. The potential field algorithm was tested against obstacles of varying shapes and sizes. Dynamic obstacles were also used, such as walking pedestrians, and the AGV was able to detect and avoid all objects within its field of view. The LRF returns were not affected by direct sunlight over the ranges of detection used by the AGV, and the system performed well in a variety of environmental conditions including direct sunlight, partial cloud cover, complete cloud cover, mist, and light fog. The chassis was able to traverse many different types of terrain, including concrete, dirt, grass, uniform mulch, and small rocks. When more difficult terrain was encountered, such as tall grass, large gravel, or loose mulch, the AGV often got stuck and was not able to reach its goal waypoint.

The system was also tested with a degraded GNSS signal by forcing the AGV under large trees and near buildings. In these cases, the GNSS fix became degraded, forcing a degradation of the position estimate. When the control algorithm detected this condition, it monitored for a good fix and reset the EKF on the GNSS/INS. The entire process happened automatically, and the AGV was able to reset its position estimate reliably every time a valid GNSS fix was lost and subsequently regained. In each of these cases, the system was able to continue toward the goal waypoint once the estimation filter was reset. The wall-following and emergency escape modes of operation were tested with both static and dynamic obstacles, and the AGV exhibited the same behavior as it did in the Gazebo simulation for each of these situations.

One of the tests completed by the AGV is illustrated in Figure 26. This figure was generated by exporting the latitude and longitude coordinates of the AGV, as reported to the ROS network, to a Keyhole Markup Language (KML) file that was then plotted in Google Earth. The AGV travelled to six waypoints on the Naval Postgraduate School campus after starting from the position indicated by the green square. The six waypoints visited—waypoints five, four, three, six, one, and two in that order—were plotted as another layer on the same overhead image.

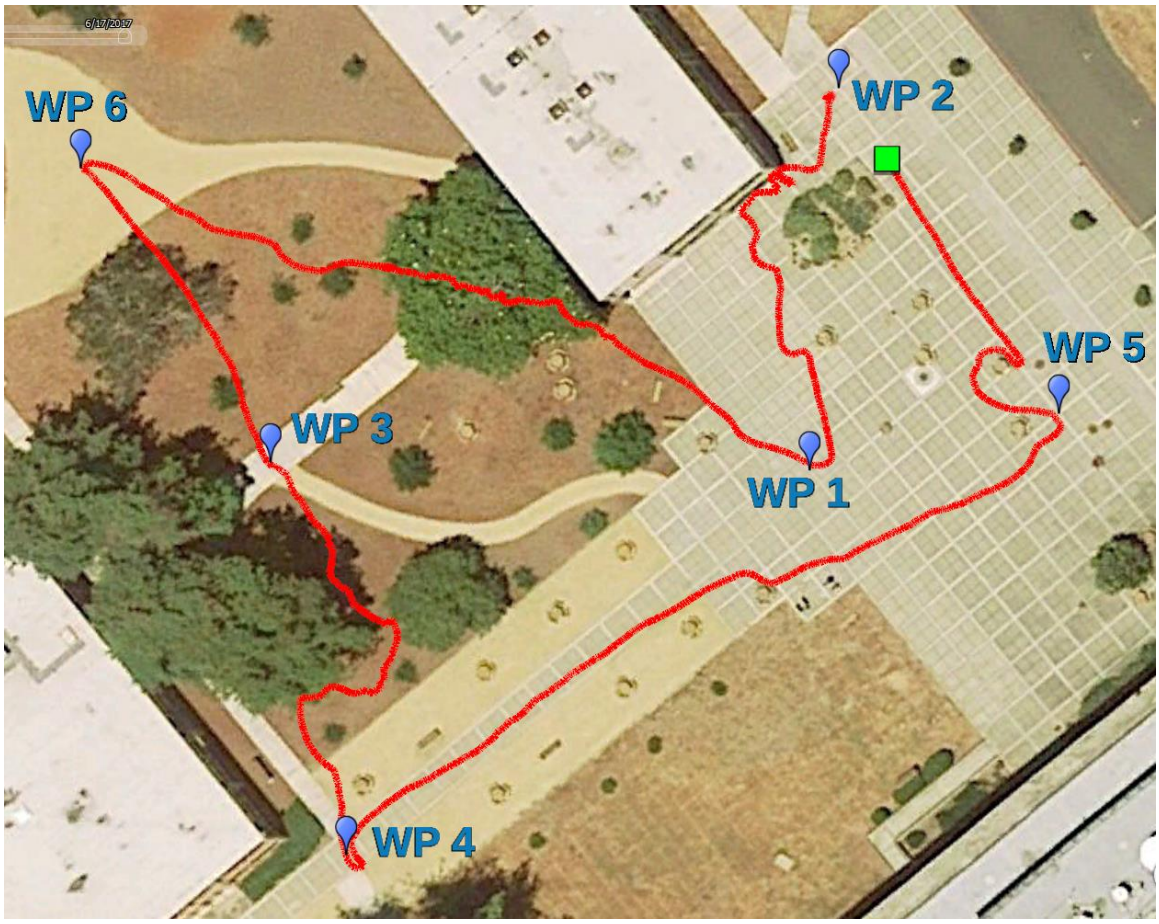

Figure 26. Outdoor AGV Test. Adapted from [39].

The AGV was able to navigate to each of the six waypoints successfully and avoided obstacles, such as planters, tables, benches, shrubs, trees, buildings, and large rocks. Near the end of this test, just before waypoint two and while the AGV was between the building and the large planter, the position estimate was degraded due to a degradation of the GNSS fix. The AGV oscillated between the building and the planter while its position estimate was degraded, as illustrated by the fluctuations in the red path in this area. Eventually, the system obtained a valid GNSS fix, and the position estimation filter was reset. Once reset, the AGV was able to continue to the final waypoint.

For the majority of this test, the system was operating in potential field mode. There was one area, however, where the AGV became trapped in a local minimum and

had to transition into the wall-following mode in order to escape. This area, near waypoint five, is illustrated in Figure 27.

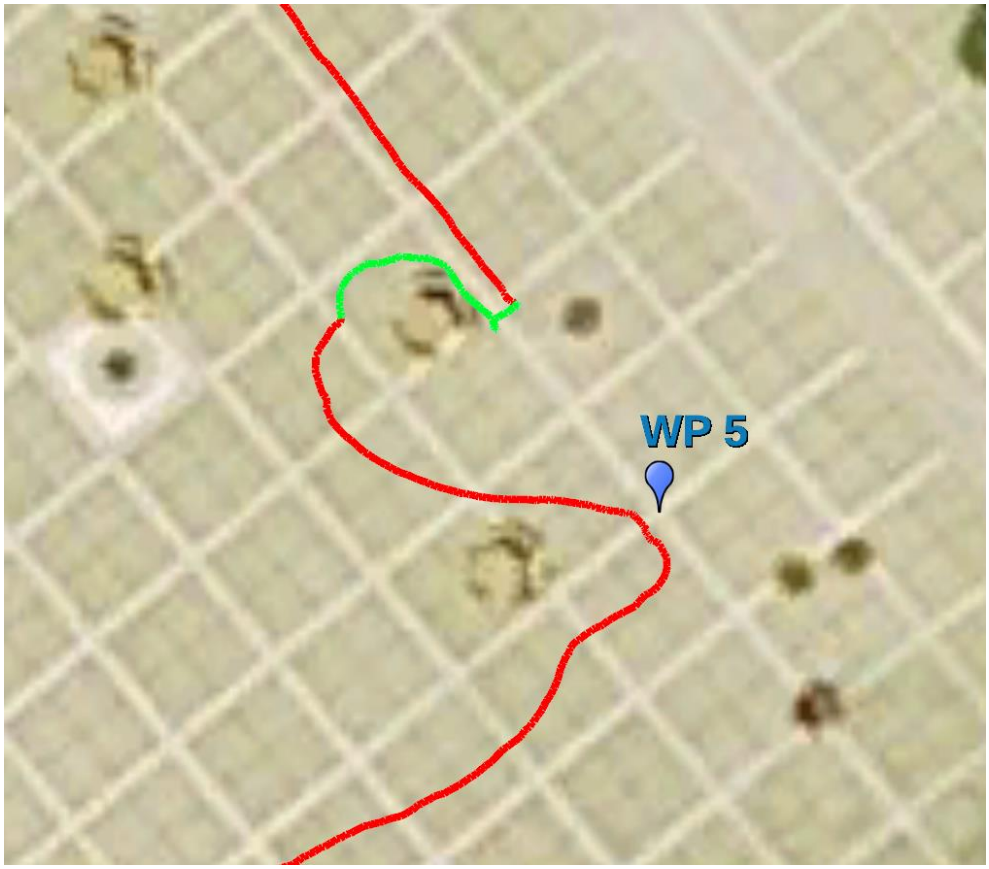

Figure 27. Outdoor AGV Test near WP 5. Adapted from [39].

Figure 27 was generated from the same Google Earth image and KML file used for Figure 26 but zoomed in to show more detail around the local minimum. The AGV encountered the local minimum generated between a planter and a table, and it transitioned into wall-following mode as indicated by the green path in Figure 27. Once the AGV was clear of the local minimum and had moved toward waypoint five again, it switched back to potential field mode and continued toward its goal.

Figures 26 and 27 are representative of the performance of the AGV during all tests performed in unstructured outdoor environments. The limitations of the chassis due to its ground clearance was observed in testing, and the system was not able to navigate

over rough or loose terrain. In some cases on rough terrain, the AGV detected false object returns from the ground as the chassis undulated over ruts and holes in the ground. When the system encountered obstacles that did not generate a LRF return, specifically due to the object's height, such as stairs, the AGV often became stuck on the obstacle. These chassis and sensor characteristics limited the environments in which the system could operate effectively, but in all other cases, the system performed well and was able to reach the target waypoint.

The outdoor testing of the AGV demonstrated that the system was able to detect and avoid obstacles to reach targeted waypoints with specific limitations. The LRF worked well and was able to detect all objects accurately within its field of view. The AGV was able to use a relatively simple navigation and control algorithm to reach its destination utilizing a minimal set of sensors. Conclusions drawn from all phases of development and testing are presented in Chapter VII, along with an assessment of how well the thesis research goals were accomplished as well as suggestions for future research and improvement.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

The research, development, and testing conducted for this thesis research resulted in an AGV that was able to successfully navigate in many different dynamic outdoor environments. While not a solution for all environments, the robot demonstrated the ability to reach a desired goal location using only a single 2D LRF for obstacle detection and avoidance. In this chapter, we provide an assessment of how well the goals of the thesis research were achieved, the limitations of the AGV, and potential areas for future work.

A. ASSESSMENT OF GOALS

The thesis research goals developed provided a method for achieving the desired purpose of the thesis research. The first goal of the thesis research—determining if the LRF is able to provide enough angular resolution and responsiveness for robust obstacle avoidance without the aid of other sensors, such as sonar—was accomplished during initial sensor testing. The data collected and analyzed, as presented in Chapter VI, resulted in the determination that the LRF was more than capable of accurately detecting almost any object within its field of view, and it was responsive enough to detect dynamic obstacles in sufficient time to facilitate avoidance. The next goal of the thesis research was to integrate sensors into an AGV for use in experimentation and testing in dynamic outdoor environments. This goal was achieved, as the robot was able to successfully navigate over varying terrain and diverse outdoor environments. Although the AGV had limitations, the system was very effective for the environments in which the chassis was designed. The final goal of the thesis research was to develop robust obstacle avoidance and navigation algorithms that were effective for both static and dynamic obstacles. This goal was also achieved, as the robot was able to successfully navigate around any obstacles within its field of view and it was able to reach the programmed goal coordinates within the accuracy of the GNSS used. The limitations of the robot design resulted in some environments and obstacles for which the robot was not able to

maneuver effectively. These issues were caused by chassis constraints and sensor limitations, however, and were not problems inherent to the guidance algorithms.

B. LIMITATIONS OF THE SYSTEM

The system developed for this thesis research was not able to operate in all outdoor environments, and it was limited primarily by the sensor used for obstacle detection. The tradeoff for utilizing a simple sensor suite was that the environment in which the robot was operating could not be completely observed by the system. In particular, the 2D geometry of the LRF's field of view limited obstacle detection to those obstacles that existed within this plane. Other obstacles, such as stairs, rocks, and potholes, would go unnoticed. In these cases, the robot was unable to reach its goal effectively unless it was able to avoid the obstacles by chance.

The other major limitation on the system, outside of extreme environments, such as water, rain, or snow, was due to the chassis design of the P3-AT. While this chassis excelled in operating on solid or moderately rough terrain, such as concrete, grass, dirt, or uniform mulch, the relatively low ground clearance limited the terrain over which it could operate. Aggravating this condition was the reduced approach and departure angles due to the bumper switches on the front and rear of the chassis. Attempting to operate the robot on rocky, rutted, or loose terrain caused the robot to become stuck, and it was not able to reach its destination.

The limitations on the AGV restricted the operation of the robot, but when operated over relatively benign terrain and with objects visible to the LRF, the system performed very well. Selecting a more capable chassis, with larger wheels and more ground clearance, would provide for a substantial improvement in performance over a wider assortment of terrain.

C. AREAS FOR FUTURE WORK

The results of this thesis research leave room for improvement to develop a more robust autonomous system. While the robot developed demonstrated the effectiveness of its obstacle avoidance and navigation algorithms, it was hindered by sensor and chassis

limitations. Without changing the sensors available on the system, different approaches could improve the AGV's overall effectiveness. Using the P3-AT's onboard sonar sensors should allow for the detection of obstacles below the plane of view of the LRF and provide visibility behind the robot when the AGV is backing up or when an object is pursuing it. Using the sonar as a secondary source of obstacle information would likely provide the robot with the ability to detect obstacles such as stairs or street curbs which otherwise would go undetected. The obstacle avoidance algorithm would be more complex in order to rectify the larger range errors present in the sonar data, and the LRF would still have to remain the primary object detection sensor, but the additional data would likely expand the types of environments for which the robot would be effective. Other improvements would be to utilize a more capable outdoor chassis for the robot, and implementing another EKF in the localization algorithm to incorporate wheel encoder data during periods of degraded GNSS fix information.

Moving beyond the sensor suite utilized for this thesis research, the biggest improvement in robot performance would be the inclusion of a LRF that scans in more than two dimensions. This could be accomplished via a gimbal system for a 2D LIDAR in order to achieve 3D data or the utilization of a dedicated 3D LIDAR scanner. By replacing the LRF used with a 3D scanner, the obstacle limitation previously described would be eliminated. This type of scanner may also be able to detect significant depressions or ruts in the ground for avoidance. The localization of the robot could be improved with additional GNSS/INS sensors to provide redundancy in the case of degraded GNSS service. The LRF could also be augmented or replaced by a camera to utilize video and image processing for obstacle and uneven terrain detection.

Replicating this sensor suite on multiple robot platforms would allow for cooperative robotic operations. Examples of these operations would be formation maneuvers, cooperative navigation through a shared environmental picture, or leader/follower operations. These types of cooperative techniques would facilitate complex operations, such as coordinated sweeps of a target area, advanced route planning, or mapping and surveying.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MATLAB CONTROL SCRIPT

```
% Pioneer 3-AT Localization and Navigation Script

% Incorporating Potential Field algorithm for navigation and GPS/IMU
% through Kalman Filter for localization

%%%% ENSURE ROS MASTER NODE IS STARTED AND MATLAB NODE GENERATED PRIOR TO
%%%% RUNNING THIS SCRIPT -- USE rosinit

%% Setup and parameter initialization

% Create global variables for use in communicating with ROS system
global Pose
global Laser
global Goal
global NavStatus
global GPSFix

% Create ROS publishers, subscribers, and service client
poseSub = rossubscriber('/geonav_p3odom',@p3atPoseCallback)
laserSub = rossubscriber('/scan',@p3atLaserCallback)
cmdPub = rospublisher('/RosAria_Node/cmd_vel','geometry_msgs/Twist')
goalPub = rospublisher('/nav/goal_odom','nav_msgs/Odometry')
casePub = rospublisher('/current_case','std_msgs/String')
goalSub = rossubscriber('/geonav_goalodom',@p3atGoalCallback)
navstatusSub = rossubscriber('/nav/status',@p3atNavStatusCallback)
fixSub = rossubscriber('/gps/fix',@p3atGPSFixCallback)
client = rossvcclient('/reset_kf')

% Pause for publisher/subscriber registration
pause(2)

% Create empty messages for publication
caseMsg = rosmessage(casePub)
cmdMsg = rosmessage(cmdPub)
goalMsg = rosmessage(goalPub)

% Get parameters and goal information the robot
[param, goals] = robotConfigReader_multigoal;

% Ask user for desired goal number
goalnum = input('Enter desired WP number (from 1 to 10):');
```

```
current_goal = goals(goalnum,:);

% Publish initial goal message for ROS system transform
for k = 1:5
   goalMsg.Pose.Pose.Position.X = current_goal(2);
   goalMsg.Pose.Pose.Position.Y = current_goal(1);
   goalMsg.Pose.Pose.Orientation.X = 0;
   goalMsg.Pose.Pose.Orientation.Y = 0;
   goalMsg.Pose.Pose.Orientation.Z = 0;
   goalMsg.Pose.Pose.Orientation.W = 1;
   send(goalPub,goalMsg);
   pause(0.1)
end

% Get current NavStatus message
navstatus = NavStatus.Data';

% Ensure NavStatus is good (2) and if not, reset KF
if navstatus(1) ~= 2
   call(client)
else
end

% Define parameters for navigation algorithm
K1 = param(3);            % forward velocity gain
K2 = param(2);            % turning velocity gain
maxvel = 3;               % maximum velocity of robot
laser_max = 20;           % robot laser view horizon
goaldist = 0.5;           % distance metric for reaching goal
goali = 1;                % current goal index
xi = param(5);            % attractive force gain
eta = param(4);           % repulsive force gain
d = param(1);             % distance above which robot velocity is constant
rho0 = param(6);          % offset from obstacle to ignore repulsive term
c = 1;                    % initial case variable
navrun = 0;               % navigation fix status variable

% Define parameters for wall-following algorithm
angK = 1;                 % turning velocity gain for WF algorithm
linK = 1;                 % forward velocity gain for WF algorithm
g_dist = [];              % initialize goal distance
g_dist0 = [];             % initialize initial goal distance
Dcount = 0;               % goal distance counter
N_Buffer = 20;            % number of measurements used to average repulsive force
Frep_Buffer = zeros(N_Buffer,1);    % initialize repulsive force buffer
```

```
 % Output velocity filter parameters
Kfilterold = 0.6;          % percentage of old velocity used
Kfilternew = 0.4;          % percentage of new velocity used
LinearVel_old = 0.0;       % initialize linear velocity
AngularVel_old = 0.0;      % initialize angular velocity

%% Potential Field Algorithm

while 1                    % Infinite loop until goal is reached
   % publish goal coordinates
   goalMsg.Pose.Pose.Position.X = current_goal(2);
   goalMsg.Pose.Pose.Position.Y = current_goal(1);
   goalMsg.Pose.Pose.Orientation.X = 0;
   goalMsg.Pose.Pose.Orientation.Y = 0;
   goalMsg.Pose.Pose.Orientation.Z = 0;
   goalMsg.Pose.Pose.Orientation.W = 1;
   send(goalPub,goalMsg);

   % get the laser ranges
   laser_range = Laser.Ranges;

   % angular resolution vector
   laser_angle = (Laser.AngleMin:Laser.AngleIncrement:Laser.AngleMax)';

   % get goal coordinates in XY world frame
   q_goal = [Goal.Pose.Pose.Position.X, Goal.Pose.Pose.Position.Y];

   % get current GPS fix
   gpsfix = [GPSFix.Status.Service,GPSFix.Status.Status]

   % get current nav status
   navstatus = NavStatus.Data'

   % if good nav status, set nav status variable
   if navstatus(1) == 2
      navrun = 1;
   else
   end

   % if bad nav status with previous good fix and good GPS fix, reset KF
   if navstatus(1) == 3 && navrun == 1 && gpsfix(2) == 30
      call(client)
      navrun = 0;
   else
   end
```

```
% switch/case for algorithm decision logic
switch c
   case 1            % Potential Field Algorithm
      fprintf('Potential Field\n')
      caseMsg.Data = 'Potential Field';   % publish current case to ROS
      send(casePub,caseMsg)

      % get X, Y and Theta
      pose = Pose.Pose.Pose;
      quat = pose.Orientation;
      angles = quat2eul([quat.W quat.X quat.Y quat.Z]);
      yaw = angles(1);
      x = pose.Position.X;
      y = pose.Position.Y;
      th = yaw;

      fprintf('X: %f, Y: %f, Theta: %f \n',x,y,th);

      % call the attractive force function
      wp_x = q_goal(goali,1);
      wp_y = q_goal(goali,2);
      [dist, angvel, linvel] = attforcepot(x,y,th,wp_x,wp_y);

      % evaluate what to do next based on the distance to the waypoint.
      if (dist <= goaldist)
         % if you have reached the goal
         if (goali < size(q_goal,1))
            % if there are multiple goals
            disp('Going to next waypoint!');
            goali = goali+1;
         else
            % if there is a single goal
            fprintf('WP #%d at x: %f, y: %f, Distance: %f\n',goalnum,wp_x,wp_y,dist);
            cmdMsg.Linear.X = 0.0;
            cmdMsg.Angular.Z = 0.0;
            fprintf('Publishing cmd_vel with lin. vel: %f, ang. vel.: %f\n', ...
               0.0,0.0);
            send(cmdPub,cmdMsg);
            disp('Done!')
            break;      % exit while loop as final goal is reached
         end
      else
         % goal not yet reached
         fprintf('WP #%d at x: %f, y: %f, Distance: %f\n',goalnum,wp_x,wp_y,dist);
         if (dist <= d)
```

```
            goalvelx = linvel;
            goalvelw = angvel;
        else
            goalvelx = maxvel;
            goalvelw = angvel;
        end
    end

    pause(0.1)          % pause for ROS system

    Frept = [0;0];      % initialize repulsive force

    for i = 1:1032
        if laser_range(i) <= laser_max
            % object position in the laser i coordinate in meters
            p_laser = [laser_range(i) 0 0 1]';
            Xobj = cos(laser_angle(i))*p_laser(1);
            Yobj = sin(laser_angle(i))*p_laser(1);
            rho = sqrt(Xobj^2+Yobj^2);
            if rho < rho0
                Frep = eta*(1/p_laser(1)-1/rho0)*(1/(p_laser(1)^2))*[-cos(laser_angle(i)) -
sin(laser_angle(i))]';
            else
                Frep = [0;0];
            end
            Frept = Frept+Frep;
        else
        end
    end

    Frep_Buffer = [Frept(2); Frep_Buffer(2:N_Buffer-1)];
    MeanBuffer = mean(Frep_Buffer);

    % calculate total force and build velocity terms
    Fatt = [goalvelx;goalvelw];
    Ftot = xi*Fatt + eta*Frept;
    fprintf('\n\nNorm of Ftot: %f\n',norm(Ftot));
    LinearVel = K1*Ftot(1);
    AngularVel = K2*Ftot(2);

    % determine which case to enter next
    if min(laser_range) < 0.5
        c = 3;
    elseif norm(Ftot) < 0.5 && dist > 1
        c = 2;
```

```
        g_dist0 = dist;
        g_dist = dist;
    else
        c = 1;
    end

case 2              % Wall-Following Algorithm
    fprintf('\nWall Following\n\n')
    caseMsg.Data = 'Wall Following';    % publish current case to ROS
    send(casePub,caseMsg)

    % get X, Y and Theta
    pose = Pose.Pose.Pose;
    quat = pose.Orientation;
    angles = quat2eul([quat.W quat.X quat.Y quat.Z]);
    yaw = angles(1);
    x = pose.Position.X;
    y = pose.Position.Y;
    th = yaw;

    fprintf('X: %f, Y: %f, Theta: %f \n',x,y,th);

    % call the attractive force function
    wp_x = q_goal(goali,1);
    wp_y = q_goal(goali,2);
    [dist, angvel, linvel] = attforcepot(x,y,th,wp_x,wp_y);
    pause(0.1)

    % if closer to the goal than last time, increment DD
    if dist < g_dist
        Dcount = Dcount + 1
    else
    end

    g_dist = dist;

    Frept = [0;0];      % initialize repulsive force

    for i = 1:1032
        if laser_range(i) <= laser_max
            % object position in the laser i coordinate in meters
            p_laser = [laser_range(i) 0 0 1]';
            Xobj = cos(laser_angle(i))*p_laser(1);
            Yobj = sin(laser_angle(i))*p_laser(1);
            rho = sqrt(Xobj^2+Yobj^2);
```

```
            if rho < rho0
               Frep = eta*(1/p_laser(1)-1/rho0)*(1/(p_laser(1)^2))*[-cos(laser_angle(i)) -
sin(laser_angle(i))]';
            else
               Frep = [0;0];
            end
            Frept = Frept+Frep;
         else
         end
      end

      % determine angle to the repulsive force vector
      objang = atan2(Frept(2),Frept(1));
      if objang < 0
         objang = objang + 2*pi;
      else
      end

      objangdeg = objang*180/pi

      % determine which way to turn and keep repulsive force vector
      % perpendicular with robot heading
      if MeanBuffer > 0
         if objangdeg >= 100
            angvel = angK*0.4;
            linvel = linK*0.05;
         elseif objangdeg < 80
            angvel = -angK*0.4;
            linvel = linK*0.05;
         else
            angvel = 0.0;
            linvel = 0.3;
         end
      elseif MeanBuffer < 0
         if objangdeg < 260
            angvel = -angK*0.4;
            linvel = linK*0.05;
         elseif objangdeg > 280
            angvel = angK*0.4;
            linvel = linK*0.05;
         else
            angvel = 0.0;
            linvel = 0.3;
         end
      end
```

```
    % develop output velocities
    LinearVel = linvel;
    AngularVel = angvel;

    % determine which case to enter next
    if min(laser_range) < 0.5
       c = 4;
    elseif Dcount == 70
       c = 1;
       g_dist = [];
       Dcount = 0;
       Frep_Buffer = zeros(N_Buffer,1);
    else
       c = 2;
    end

case 3          % Emergency Avoidance Algorithm (From Potential Field)
    ii = 0;
    while ii < 5
       % stop immediately for 5 seconds
       fprintf('Emergency Avoidance\n')
       caseMsg.Data = 'Emergency Avoidance (PF)';
       send(casePub,caseMsg)
       % populate the message
       fprintf('WP #%d at x: %f, y: %f, Distance: %f\n',goalnum,wp_x,wp_y,dist);
       cmdMsg.Linear.X = 0.0;
       cmdMsg.Angular.Z = 0.0;
       % publish message
       fprintf('Publishing cmd_vel with lin. vel: %f, ang. vel.: %f\n', ...
          0.0,0.0);
       send(cmdPub,cmdMsg);
       pause(0.2)
       ii = ii + 0.2;
    end
    jj = 0;
    while jj < 4
       % backup for 4 seconds to make enough room to maneuver
       % around obstacle
       caseMsg.Data = 'Emergency Avoidance (PF)';
       send(casePub,caseMsg)
       fprintf('WP #%d at x: %f, y: %f, Distance: %f\n',goalnum,wp_x,wp_y,dist);
       cmdMsg.Linear.X = -0.2;
       cmdMsg.Angular.Z = 0.0;
       % publish
       fprintf('Publishing cmd_vel with lin. vel: %f, ang. vel.: %f\n', ...
```

```
            0.0,0.0);
        send(cmdPub,cmdMsg);
        pause(0.2);
        jj = jj + 0.2;
    end

    % get the laser ranges
    laser_range = Laser.Ranges;

    % determine if obstacle is out of minimum range parameter
    if min(laser_range) < 0.5
        c = 3;
    else
        c = 1;
    end

case 4              % Emergency Avoidance Algorithm (From Wall Following)
    ii = 0;
    while ii < 5
        % stop immediately for 5 seconds
        fprintf('Emergency Avoidance\n')
        caseMsg.Data = 'Emergency Avoidance (WF)';
        send(casePub,caseMsg)
        % populate the twist message
        fprintf('WP #%d at x: %f, y: %f, Distance: %f\n',goalnum,wp_x,wp_y,dist);
        cmdMsg.Linear.X = 0.0;
        cmdMsg.Angular.Z = 0.0;
        % publish
        fprintf('Publishing cmd_vel with lin. vel: %f, ang. vel.: %f\n', ...
            0.0,0.0);
        send(cmdPub,cmdMsg);
        pause(0.2)
        ii = ii + 0.2;
    end
    jj = 0;
    while jj < 4
        % backup for 4 seconds to make enough room to maneuver
        % around obstacle
        caseMsg.Data = 'Emergency Avoidance (WF)';
        send(casePub,caseMsg)
        fprintf('WP #%d at x: %f, y: %f, Distance: %f\n',goalnum,wp_x,wp_y,dist);
        cmdMsg.Linear.X = -0.2;
        cmdMsg.Angular.Z = 0.0;
        % publish
        fprintf('Publishing cmd_vel with lin. vel: %f, ang. vel.: %f\n', ...
```

```
            0.0,0.0);
        send(cmdPub,cmdMsg);
        pause(0.2);
        jj = jj + 0.2;
      end

      % get the laser ranges
      laser_range = Laser.Ranges;

      % determine if obstacle is out of minimum range parameter
      if min(laser_range) < 0.5
        c = 3;
      else
        c = 2;
      end

    otherwise
  end

  % build filtered output velocity parameters
  cmdMsg.Linear.X = Kfilternew*LinearVel + Kfilterold*LinearVel_old;
  cmdMsg.Angular.Z = Kfilternew*AngularVel + Kfilterold*AngularVel_old;

  % publish on cmd_vel topic
  fprintf('Publishing cmd_vel with lin. vel: %f, ang. vel.: %f\n', ...
    cmdMsg.Linear.X,cmdMsg.Angular.Z);
  send(cmdPub,cmdMsg);

  LinearVel_old = cmdMsg.Linear.X;
  AngularVel_old = cmdMsg.Angular.Z;

end
```

APPENDIX B. ROS LAUNCH FILE

```
<launch>

<!-- Startup the RosAria node -->
  <node pkg="rosaria" name="RosAria_Node" type="RosAria" output="screen">
  <param name="/port" value="/dev/ttyS0"/>
  </node>

<!-- Startup the Hokuyo node -->
  <node name="Laser_Node" pkg="hokuyo_node" type="hokuyo_node"
respawn="false" output="screen">
    <param name="min_ang" value="-2.25"/>
    <param name="max_ang" value="2.25"/>

    <!-- Starts up faster, but timestamps will be inaccurate. -->
    <param name="calibrate_time" type="bool" value="true"/>

    <!-- Set the port to connect to here -->
    <param name="port" type="string" value="/dev/Hokuyo"/>

    <param name="intensity" type="bool" value="false"/>
  </node>

<!-- Startup the USB Webcam node -->
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen"
respawn="true">
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1280" />
    <param name="image_height" value="720" />
    <param name="pixel_format" value="yuyv" />
    <param name="framerate" value="30" />
    <param name="autofocus" value="true" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <!--node name="image_view" pkg="image_view" type="image_view"
respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node-->

<!-- Startup the Microstain sensor node -->
  <node name="microstrain_3dm_gx5_45_node"
```

```
  pkg="microstrain_3dm_gx5_45"
  type="microstrain_3dm_gx5_45_node" output="screen" respawn="true">
    <!--<param name="port" value="/dev/Microstrain45" type="str" />-->
    <param name="port" value="/dev/Microstrain45" type="str" />
    <param name="baudrate" value="115200" type="int" />

    <param name="device_setup" value="true" type="bool" />
    <param name="readback_settings" value="true" type="bool" />
    <param name="save_settings" value="true" type="bool" />
    <param name="auto_init" value="true" type="bool" />
    <param name="gps_rate" value="4" type="int" />
    <param name="imu_rate" value="10" type="int" />
    <param name="nav_rate" value="10" type="int" />
    <param name="dynamics_mode" value="1" type="int" />
    <param name="declination_source" value="2" type="int" />
    <param name="declination" value="0.23" type="double" />

    <param name="gps_frame_id" value="wgs84" type="str" />
    <param name="imu_frame_id" value="base_link" type="str" />
    <param name="odom_frame_id" value="wgs84" type="str" />
    <param name="odom_child_frame_id" value="base_link" type="str" />

    <param name="publish_gps" value="true" type="bool" />
    <param name="publish_imu" value="true" type="bool" />
    <param name="publish_odom" value="true" type="bool" />
  </node>

<!-- Startup the geonav transform node -->
  <node pkg="geonav_transform" type="geonav_transform_node"
name="geonav_transform_node" clear_params="true" output="screen">
    <!-- Datum as latitude, longitude [decimal deg.], yaw [ENU, degrees] -->
    <rosparam param="datum">[36.5952165660384, -121.875074147324,
0.0]</rosparam>
    <remap from="nav_odom" to="nav/odom"/>
    <remap from="geonav_odom" to="geonav_p3odom"/>
    <remap from="geonav_utm" to="geonav_p3utm"/>
  </node>

<!-- Startup the goal geonav transform node -->
  <node pkg="geonav_transform" type="goal_geonav_transform_node"
name="goal_geonav_transform_node" clear_params="true" output="screen">
    <!-- Datum as latitude, longitude [decimal deg.], yaw [ENU, degrees] -->
    <rosparam param="datum">[36.5952165660384, -121.875074147324,
0.0]</rosparam>
    <remap from="nav/odom" to="nav/goal_odom"/>
```

```
    <remap from="geonav_odom" to="geonav_goalodom"/>
    <remap from="geonav_utm" to="geonav_goalutm"/>
  </node>

</launch>
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] J. Larson and M. Trivedi, "Lidar based off-road negative obstacle detection and analysis," in *14th Int. IEEE Conf. on Intell. Transp. Syst.*, Washington, DC, 2011, pp. 192–197.

[2] I. Shim, D. G. Choi, S. Shin, and I. S. Kweon, "Multi lidar system for fast obstacle detection," in *9th Int. Conf. on Ubiquitous Robots and Ambient Intell.*, Daejeon, South Korea, 2012, pp. 557–558.

[3] D. F. Wolf and C. N. S. Netto, "Vision-based outdoor navigation using mobile robots," in *IEEE Latin American Robotic Symp.*, Natal, Rio Grande do Norte, Brazil, 2008, pp. 35–39.

[4] A. Cherubini, F. Spindler, and F. Chaumette, "Autonomous visual navigation and laser-based moving obstacle avoidance," *IEEE Trans. on Intell. Transp. Syst.*, vol. 15, no. 5, pp. 2101–2110, April 2, 2014.

[5] G. Zhao, X. Xiao, and J. Yuan, "Fusion of Velodyne and camera data for scene parsing," in *15th Int. Conf. on Inform. Fusion*, Singapore, 2012, pp. 1172–1179.

[6] A. Saleem, A. Al Maashri, L. Khriji, and M. Hussein, "An integration framework for UGV outdoor navigation system based on lidar and vision data," in *16th Int. Conf. on Res. and Edu. in Mechatronics*, Bochum, Germany, 2015, pp. 16–21.

[7] S. Hu, C. Chen, A. Zhang, W. Sun, and L. Zhu, "A small and lightweight autonomous laser mapping system without GPS," *J. of Field Robotics*, vol. 30, no. 5, pp. 784–802, September 2013. [Online]. doi:10.1002/rob.21465

[8] E. Shang, X. An, J. Li, and H. He, "A novel setup method of 3D lidar for negative obstacle detection in field environment," in *17th Int. IEEE Conf. on Intell. Transp. Syst.*, Qingdao, China, 2014, pp. 1436–1441.

[9] F. J. Botha, C. E. van Daalen, and J. Treurnicht, "Data fusion of radar and stereo vision for detection and tracking of moving objects," in *Pattern Recognition Assoc. of South Africa and Robotics and Mechatronics Int. Conf.*, Stellenbosch, South Africa, 2016, pp. 1–7.

[10] S. A. Rodriguez F., V. Fremont, P. Bonnifait, and V. Cherfaoui, "An embedded multi-modal system for object localization and tracking," *IEEE Intell. Transp. Syst. Mag.*, vol. 4, no. 4, pp. 42–53, November 12, 2012.

[11] M. A. Ansari and F. A. Umrani, "Sonar based obstacle detection and avoidance algorithm," in *Int. Conf. on Signal Acquisition and Process.*, Kuala Lumpur, 2009, pp. 98–102.

[12] “Pioneer 3-AT,” Omron Adept MobileRobots. Accessed October 11, 2017. [Online]. Available: http://www.mobilerobots.com/ResearchRobots/P3AT.aspx

[13] “P3-AT datasheet,” Adept MobileRobots. Accessed October 11, 2017. [Online]. Available: http://www.mobilerobots.com/Libraries/Downloads/Pioneer3AT-P3AT-RevA.sflb.ashx

[14] “Research robots specifications,” Omron Adept MobileRobots. Accessed October 11, 2017. [Online]. Available: http://www.mobilerobots.com/ResearchRobots/ResearchMatrix.aspx

[15] Adept MobileRobots, *Pioneer 3 Operations Manual*, ver. 6.2, Amherst, NH, USA: Adept MobileRobots, 2011.

[16] “UTM-30LX,” Hokuyo Automatic Co. Accessed October 11, 2017. [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=169

[17] “Scanning laser range finder UTM-30LX/LN specification,” Hokuyo Automatic Co., November 27, 2012. [Online]. Available: http://www.robotshop.com/media/files/pdf/hok-06-sepcs.pdf

[18] “3DM-GX5-45 datasheet,” LORD MicroStrain. Accessed October 11, 2017. [Online]. Available: http://www.microstrain.com/sites/default/files/3dm-gx5-45_datasheet_8400-0091.pdf

[19] “3DM-GX5-45 GNSS-aided inertial navigation system,” LORD MicroStrain. Accessed October 11, 2017. [Online]. Available: http://www.microstrain.com/inertial/3dm-gx5-45

[20] “SlimPRO SP675P mini PC,” CappuccinoPC. Accessed October 11, 2017. [Online]. Available: http://www.cappuccinopc.com/slimpro-sp675p.asp

[21] “About ROS,” Open Source Robotics Foundation. Accessed October 11, 2017. [Online]. Available: http://www.ros.org/about-ros/

[22] “Hokuyo_node,” Open Source Robotics Foundation, September 8, 2016. [Online]. Available: http://wiki.ros.org/hokuyo_node

[23] “Microstrain_3dm_gx5_45,” Open Source Robotics Foundation, April 14, 2017. [Online]. Available: http://wiki.ros.org/microstrain_3dm_gx5_45

[24] “ROSARIA,” Open Source Robotics Foundation, March 1, 2017. [Online]. Available: http://wiki.ros.org/ROSARIA

[25] “ARIA,” Omron Adept MobileRobots, December 13, 2016. [Online]. Available: http://robots.mobilerobots.com/wiki/ARIA

[26] “Usb_cam,” Open Source Robotics Foundation, March 5, 2016. [Online]. Available: http://wiki.ros.org/usb_cam

[27] “MATLAB overview,” MathWorks. Accessed October 11, 2017. [Online]. Available: https://www.mathworks.com/products/matlab.html

[28] “Robot operating system (ROS) support from robotics system toolbox,” MathWorks. Accessed October 11, 2017. [Online]. Available: https://www.mathworks.com/hardware-support/robot-operating-system.html

[29] “Gazebo,” Open Source Robotics Foundation. Accessed October 11, 2017. [Online]. Available: http://gazebosim.org/

[30] “Beginner: overview,” Open Source Robotics Foundation. Accessed October 11, 2017. [Online]. Available: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1

[31] “ROS overview,” Open Source Robotics Foundation. Accessed October 11, 2017. [Online]. Available: http://gazebosim.org/tutorials/?tut=ros_overview

[32] “Gazebo_ros_pkgs,” Open Source Robotics Foundation, May 24, 2017. [Online]. Available: http://wiki.ros.org/gazebo_ros_pkgs

[33] D. Wonnacott, “Pioneer3at,” Open Source Robotics Foundation, May 9, 2017. [Online]. Available: http://models.gazebosim.org/pioneer3at/

[34] J. Hsu, “Hokuyo,” Open Source Robotics Foundation, May 9, 2017. [Online]. Available: http://models.gazebosim.org/hokuyo/

[35] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS*. Sebastopol, CA, USA: O’Reilly Media, Inc., 2015.

[36] J. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[37] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA, USA: MIT Press, 2005.

[38] E. R. Bachmann, X. Yun, and A. Brumfield, “Limitations of attitude estimation algorithms for inertial/magnetic sensor modules,” *IEEE Robot. and Autom. Mag.*, vol. 14, no. 3, pp. 76–87, September 2007.

[39] Google Earth Pro 7.3.0.3832. (June 17, 2017). “Naval Postgraduate School.” 36.595°N, 121.875°W, Eye alt 131 m. Accessed October 25, 2017.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California